作业4 SIMD实验

Exercise 1：熟悉 SIMD intrinsics 函数

- Four floating point divisions in single precision (i.e. float)（4 个并行的单精度浮点数除法）

```
__m128 _mm_div_ps(__m128 a, __m128 b)
```

- Sixteen max operations over unsigned 8-bit integers (i.e. char) （16 个并行求 8 位无符号整数的最大值）

```
__m128i _mm_max_epu8(__m128i a, __m128i b)
```

- Arithmetic shift right of eight signed 16-bit integers (i.e. short) （8 个并行的 16 位带符号短整数的算术右移）

```
__m128i _mm_srli_epi16(__m128i a, int imm8)
```

Exercise 2：阅读 SIMD 代码

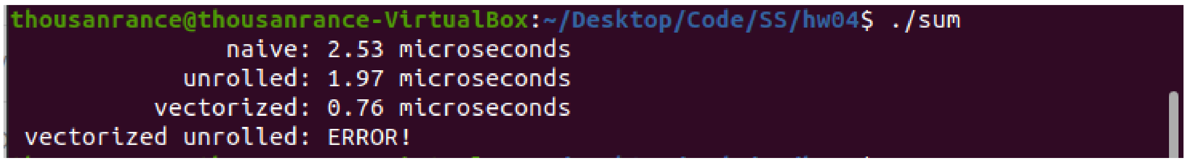
- sseTest.s 文件的内容， 哪些指令是执行 SIMD 操作的?

```
porx
movsd
movapd
addpd
mulpd
movaps
unpckhpd
```

Exercise 3：书写 SIMD 代码

```
static int sum_vectorized(int n, int *a)
{
    // WRITE YOUR VECTORIZED CODE HERE

    __m128i sum0_3 = _mm_setzero_si128(); //[0, 0, 0, 0]

    for(int i = 0; i < n/4; i++)
    {
        __m128i addi0_3 = _mm_loadu_si128(a + i*4); //[a_(4i+0), a_(4i+1),
a_(4i+2), a_(4i+3)]

        sum0_3 = _mm_add_epi32(sum0_3, addi0_3);
    }

    int sum_vec[4];
    _mm_storeu_si128(sum_vec, sum0_3);

    int sum = 0;
    for(int i = 0; i < 4; i++)
    {
        sum += sum_vec[i];
    }

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

    return sum;
}
```

- 运行结果：



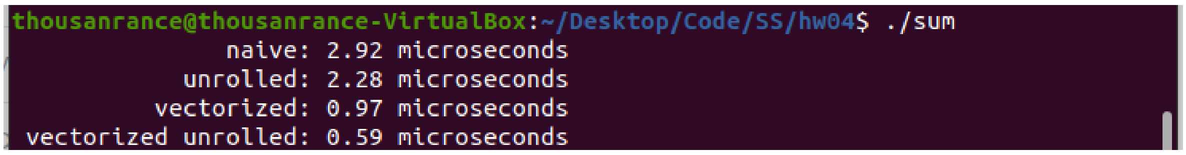
```
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/SS/hw04$ ./sum
               naive: 2.53 microseconds
            unrolled: 1.97 microseconds
          vectorized: 0.76 microseconds
 vectorized unrolled: ERROR!
```

性能有所改善。

Exercise 4：Loop Unrolling 循环展开

```
static int sum_vectorized_unrolled(int n, int *a)
{
    // UNROLL YOUR VECTORIZED CODE HERE

    __m128i sum0_3 = _mm_setzero_si128(); //[0, 0, 0, 0]

    for(int i = 0; i < n/16; i++)
    {
        __m128i addi0_3 = _mm_loadu_si128(a + i*16);
        sum0_3 = _mm_add_epi32(sum0_3, addi0_3);

        addi0_3 = _mm_loadu_si128(a + i*16 + 4);
        sum0_3 = _mm_add_epi32(sum0_3, addi0_3);

        addi0_3 = _mm_loadu_si128(a + i*16 + 8);
        sum0_3 = _mm_add_epi32(sum0_3, addi0_3);

        addi0_3 = _mm_loadu_si128(a + i*16 + 12);
        sum0_3 = _mm_add_epi32(sum0_3, addi0_3);

    }

    int sum_vec[4];
    _mm_storeu_si128(sum_vec, sum0_3);

    int sum = 0;
    for(int i = 0; i < 4; i++)
    {
        sum += sum_vec[i];
    }
    for(int i = n/16 * 16; i < n; i++)
    {
        sum += a[i];
    }

    return sum;
}
```

- 运行结果：



```
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/SS/hw04$ ./sum
               naive: 2.92 microseconds
            unrolled: 2.28 microseconds
          vectorized: 0.97 microseconds
 vectorized unrolled: 0.59 microseconds
```

性能有所改善。

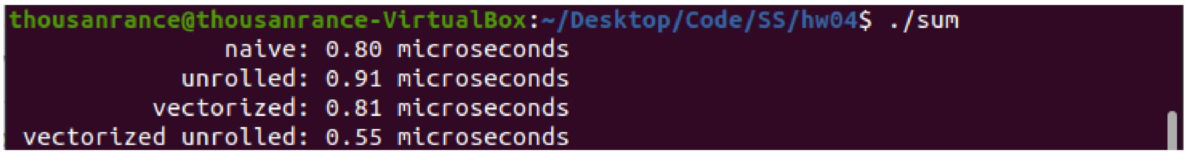
Exercise 5

采用 gcc -o3 自动向量化

- 修改Makefile文件：

```
FLAGS = -std=gnu99 -O3 -DNDEBUG -g0 -msse4.2
```

- 运行结果：



```
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/SS/hw04$ ./sum
               naive: 0.80 microseconds
            unrolled: 0.91 microseconds
          vectorized: 0.81 microseconds
 vectorized unrolled: 0.55 microseconds
```

性能有所改善，改善后的性能与手动向量化后一样。

采用 -mavx2 编译选项

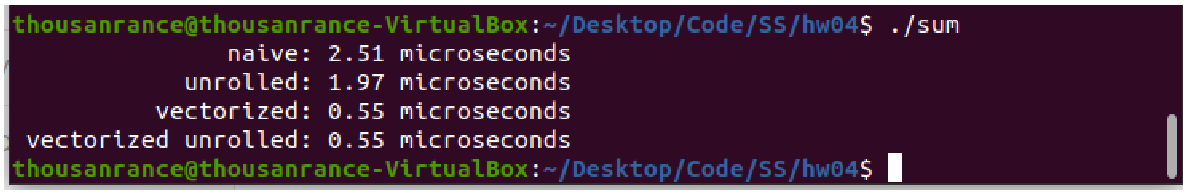
- 修改Makefile文件：

```
FLAGS = -std=gnu99 -O2 -DNDEBUG -g0 -mavx2
```

- 修改sum_vectorized()函数为：

```
static int sum_vectorized(int n, int *a)
{
    // WRITE YOUR VECTORIZED CODE HERE

    __m256i sum0_7 = _mm256_setzero_si256();

    for(int i = 0; i < n/8; i++)
    {
        __m256i addi0_7 = _mm256_loadu_si256(a + i*8);

        sum0_7 = _mm256_add_epi32(sum0_7, addi0_7);
    }

    int sum_vec[8];
    _mm256_storeu_si256(sum_vec, sum0_7);

    int sum = 0;
    for(int i = 0; i < 8; i++)
    {
        sum += sum_vec[i];
    }

    for(int i = n/8 * 8; i < n; i++)
    {
        sum += a[i];
    }

    return sum;
}
```

- 运行结果：



```
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/SS/hw04$ ./sum
               naive: 2.51 microseconds
            unrolled: 1.97 microseconds
          vectorized: 0.55 microseconds
 vectorized unrolled: 0.55 microseconds
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/SS/hw04$ 
```

性能有所改善，改善后的性能与循环展开后一样。

通过以上实验可以看出，指令的并行度越高，程序性能越好。