Lab5 - Intel SIMD

UNIkeEN

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

通过阅读代码与查询手册，可得执行SIMD操作的指令有：

```
movsd, movapd, mulpd, addpd, movaps, unpckhpd
```

Exercise 3: 书写 SIMD 代码

代码如下：

```
static int sum_vectorized(int n, int *a)
{
    int sum = 0;
    int temp[4] = {0,0,0,0};
    __m128i sum_vec = _mm_setzero_si128();

    // vectorized loop
    for (int i = 0; i < n / 4 * 4; i += 4)
    {
        __m128i addend = _mm_loadu_si128((__m128i *)(a + i));
        sum_vec = _mm_add_epi32(sum_vec, addend);
    }

    _mm_storeu_si128((__m128i *)temp, sum_vec);
    sum = temp[0] + temp[1] + temp[2] + temp[3];

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

运行结果如下：

```
               naive: 3.87 microseconds
            unrolled: 3.33 microseconds
          vectorized: 1.74 microseconds
 vectorized unrolled: 0.74 microseconds
```

可以发现性能得到改善，`vectorized` 版本的执行时间缩短至 `naive` 版本的 1/2 以下。

Exercise 4: Loop Unrolling 循环展开

仿照`unrolled`中的代码，做四路循环展开如下：

```c
static int sum_vectorized_unrolled(int n, int *a)
{

    int sum = 0;
    int temp[4] = {0,0,0,0};
    __m128i sum_vec = _mm_setzero_si128();

    // unrolled vectorized loop
    for (int i = 0; i < n / 16 * 16; i += 16)
    {
        __m128i addend0 = _mm_loadu_si128((__m128i *)(a + i));
        __m128i addend1 = _mm_loadu_si128((__m128i *)(a + i + 4));
        __m128i addend2 = _mm_loadu_si128((__m128i *)(a + i + 8));
        __m128i addend3 = _mm_loadu_si128((__m128i *)(a + i + 12));
        sum_vec = _mm_add_epi32(sum_vec, addend0);
        sum_vec = _mm_add_epi32(sum_vec, addend1);
        sum_vec = _mm_add_epi32(sum_vec, addend2);
        sum_vec = _mm_add_epi32(sum_vec, addend3);
    }

    _mm_storeu_si128((__m128i *)temp, sum_vec);
    sum = temp[0] + temp[1] + temp[2] + temp[3];

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

输出结果见 **Exercise3**，可以看到性能再次提升约一倍。