第四次作业

学号	姓名
20319045	刘冠麟

第一题

浮点型数据以类似科学计数法的方式存储，而且由于浮点变量能存储小数点后3位数字，所以数组a为：

```
a[] = {4.000e+00,  3.000e+00, 3.000e+00, 1.000e+03}
```

程序将i=0，1的迭代结果分配给线程0，i=2，3的迭代结果分配给线程1。系统为每个线程创建一个私有变量，每个线程的私有变量各自计算自己的和：

- 线程0计算4.000e+00， 3.000e+00的和，计算结果存储在线程私有变量`sum0`中，计算结果为7.000e+00
- 线程1计算3.000e+00， 1.000e+03的和，因为寄存器能保存小数点后四位，所以寄存器中的计算结果为1.0030e+03，存储到线程私有变量`sum1`中时要四舍五入，所以计算结果为1.003e+03

然后线程0和线程1将各自变量相加，相加在寄存器中的结果为1.0100e+03，四舍五入到总的计算结果变量中为1.010e+03，最后由于程序输出为`printf ( "sum=%4.1f \n" .sum) ;`，所以输出结果为：**1010.0**

第二题

（1）程序中存在的循环依赖：由于计算的每一个`a[i]`都要通过`a[i-1]`计算得出，所以想要得出`a[i]`必须先计算出前面i-1个a[i]的值。因此难以并行化。

但是由于这是一个**递增等差数列**，可以写出程序的公式直接由`i`推导出a_i如下：

$$a_i = i$$

可以重写循环为：

```
a [ o ] = 0;
for ( i =1; i < n ; i++)
    a [ i ] = i;
```

（2）加入openmp并行化：

```
# pragma omp parallel for num_threads(thread_count) \ default(none) private(i)
shared(a, n)
for(i=0;i<n;i++)
    a[i]= i;
```

第三题

缓存行大小为 64 字节，每个 double 类型数据占用 8 字节。因此，每个缓存行可以存储 64 / 8 = 8 个 double 类型数据。

根据题目有四个线程，线程具体分配如下

- **线程 0** 处理 y 的元素从 y[0] 到 y[1999]。
- **线程 1** 处理 y 的元素从 y[2000] 到 y[3999]。
- **线程 2** 处理 y 的元素从 y[4000] 到 y[5999]。
- **线程 3** 处理 y 的元素从 y[6000] 到 y[7999]。

线程 0 和 线程 2

线程0 处理的元素范围是 y[0] 到 y[1999]，线程2处理的元素范围是 y[4000]到 y[5999]，线程 0 的最后一个元素是 y[1999]，而线程 2 的第一个元素是 y[4000]。

给出缓存行可以包含元素，线程0处理的最后一个元素 y[1999] 和线程2处理的第一个元素 y[4000]在索引上相差2001，远大于一个缓存行所能包含的元素数（8个double）。因此，**线程0和线程2之间不会发生伪共享。**

线程 0 和 线程 3

线程 3 的第一个元素是 y[6000]。同样根据元素索引，线程 0 的缓存行不可能与线程 3 的缓存行重叠。**线程 0 和线程 3 之间也不会发生伪共享。**

第四题

```
void matvec_mult(float *A, float *x, float *y, int m, int n, int thread_count) {
    #pragma omp parallel for num_threads(thread_count) default(none) private(i)
shared(A, x, y, n, m)
    for (int i = 0; i < m; i++) {
        float temp = 0.0;
        for (int j = 0; j < n; j++) {
            temp += A[i * n + j] * x[j];
        }
        y[i] = temp;
    }
}
```