超级计算机原理与操作 Homework 5

习题 6.1

在串行 n 体问题算法的每一次迭代中，首先计算每一个粒子所受的总作用力，然后为每一个粒子计算他们的位置和速度。是否可以重新组织代码，使得在每一次迭代中，在进入下一个粒子的计算前，能够先完成对一个粒子的所有计算？如果是，可否采用如下的伪代码：（请说明理由）

```
for each timestep
   for each particle {
      Compute total force on particle;
      Find position and velocity of particle;
      Print position and velocity of particle;
   }
```

习题 6.4

用 OpenMP 或者 Pthreads 来并行化 n 体问题的简化版本算法，并且对每一个粒子使用一个锁/互斥量。锁/互斥量是用来对 force 数组的更新试试保护的。通过并行化内层 for 循环来并行化解法的剩下部分。与串行算法相比，这个代码的性能如果？解释你的回答。

习题 6.8

x 和 y 都是双精度的 n 为向量，而α是一个双精度标量，我们称以下赋值：

$$y := \alpha x + y$$

为双精度的α乘以 x 再加上 y(Double precision Alpha times X Plus Y,DAXPY)

编写一个 OpenMP 或者 Pthreads 程序，在主线程中生成两个随机 n 为大数字，一个随机标量，他们的类型都是 double。然后由多个线程对随机生成的数据执行 DAXPY。用大数值的 n 和不同数目的线程来运行程序，比较分别采用块划分和循环划分对数组进行划分时，哪一种划分策略的性能较好？为什么？

习题 6.10

考虑以下程序

```
int n, thread_count, i, chunksize;
double x[n], y[n], a;
    . . .
#   pragma omp parallel num_threads(thread_count) \
       default(none) private(i) \
       shared(x, y, a, n, thread_count, chunksize)
     {
 #     pragma omp for schedule(static, n/thread_count)
       for (i = 0; i < n; i++) {
          x[i] = f(i);  /* f is a function */
          y[i] = g(i);  /* g is a function */
       }
 #     pragma omp for schedule(static, chunksize)
       for (i = 0; i < n; i++)
          y[i] += a*x[i];
    }  /*  omp parallel */
```

假定 n = 64, thread_count = 2, 高速缓存的大小是 8 个 double，每一个核有一个能存储 131072 个 double 型数据的二级（L2）高速缓存。如果 chunksize = n / thread_count, 那么在第二个循环中会出现多少次 L2 高速缓存缺失？ 你可以假定 x 和 y 都按照高速缓存行的边界对齐存储。即 x[0]和 y[0]都是他们所

在的高速缓存行的第一个元素。

习题 6.18

广度优先搜索可以实现为一个迭代算法，其中要采用一个队列。队列是“先入先出”的链表数据结构，队列中的元素出队的顺序与入队的顺序是一致的。可以用队列来解决 TSP 问题，广度优先搜索的实现如下：

```
queue = Init_queue();  /* Create empty queue */
tour = Init_tour();    /* Create partial tour that visits
    hometown */
Enqueue(queue, tour);
while (!Empty(queue)) {
   tour = Dequeue(queue);
   if (City_count(tour) == n) {
      if (Best_tour(tour))
         Update_best_tour(tour);
   } else {
      for each neighboring city
         if (Feasible(tour, city)) {
            Add_city(tour, city);
            Enqueue(tour);
            Remove_last_city(tour);
         }
   }
   Free_tour(tour);
}  /* while !Empty */
```

这个算法虽然是正确的，但当城市数目超过 10 时，实现起来相当困难。为什么？在 TSP 问题的共享内存解决方案中，可以使用广度优先搜索来创建初始回路列表，这些回路能分配给各个线程。

a. 修改上述代码，以便线程 0 可以用此代码来生成数目至少为 thread_count 的回路队列。

b. 一旦线程 0 生成了回路队列，编写出各个线程如何用队列里的一部分回路来初始化自己栈的伪代码。