1、答：

（1）Filter Lock 算法：

```
int level[p];
int victim[p]; //use 1..n-1
for(int i = 0; i < p; ++i)
{
    level[i] = 0;
}

void lock(int tid)
{
    for (int i=1; i<p; i++)
    {
        level[tid] = 1;
        victim[i] = tid;
        while (same_or_higher(tid, i) && victim[i]==tid);
    }
}

void unlock(int tid)
{
    level[tid] = 0;
}

bool same_or_higher(int tid, int i)
{
    for (int k=0; k<p; k++)
        if (k!=tid && level[k] >= i) return true;
    return false;
}
```

（2）Lamport's bakery 算法：

```
int number[p];
bool entering[p];
for(int i = 0; i < p; ++i)
{
    number[i] = 0;
    entering[i] = false;
}

void lock(int tid)
{
    entering[tid] = true;
    number[tid] = 1 + array_max(number); //find max in the array
    entering[tid] = false;
    for (int i=0; i<p; i++)
    {
        if (i != tid)
        {
            // Wait until thread i receives its number
            while (entering[i]);
            // Wait until all threads with smaller numbers
            // or with the same number, but with higher priority,
            // finish their work
            while ((number[i]!=0) && (number[tid]>number[i] ||
                    (number[tid]==number[i] && tid>i)));
        }
    }
}

void unlock(int tid)
{
    number[tid] = 0;
}
```

（3）最少需要访问 p 个内存地址。假设只需要 i 个地址，存在某个时刻，p 个进程都要向这 i 个地址写以表明自己想获得锁。如果 i<p, 必然有某个进程 $p_j$ 希望获得锁的意图会被其它进程的写覆盖掉，从而其它进程不知道 $p_j$ 希望获得锁。这样就会产生不公平。

2、答：

（1）Compare_And_Swap(CAS)和 Load-Linked and Store-Conditional(LL/SC)是两种比较常见的硬件同步原语。例如，Intel、AMD 的 x86 指令集和 Oracle/SUN 的 Sparc 指令集实现了 CAS 指令；Alpha、PowerPC、MIPS、ARM 均实现了 LL/SC 指令。

（2）CAS 指令硬件实现比 LL/SC 的硬件实现复杂。使用 CAS 指令会碰到 ABA 问题，但是 LL/SC 指令不会碰到该问题。LL/SC 指令中由于 SC 是尝试去写，因此在某些情况下，SC 执行成功率很低，导致用 LL/SC 实现的锁执行开销变得很大。

（3）下面仅给出用 CAS 和 LL/SC 实现普通自旋锁的例子。公平的自旋锁可利用这些普通的自旋锁实现。

CAS：指令定义 cas r1, r2, Mem；r1 存放期望值，r2 存放更新值，

```
获得锁：//锁初始化为 0
    la      t0, sem
TryAgain:
    li      t1, 0
    li      t2, 1
    cas     t1, t2, 0x0(t0)
    bnez    t1, TryAgain
    nop
释放锁：
    la      t0, sem
    li      t1, 0
    sw      t1, 0x0(t0)
```

LL/SC：

```
获得锁：//锁初始化为 0
    la      t0, sem
TryAgain:
    ll      t1, 0x0(t0)
    bnez    t1, TryAgain
    li      t1, 1
    sc      t1, 0x0(t0)
    beqz    t1, TryAgain
    nop
释放锁：
    la      t0, sem
    sw      zero, 0x0(t0)
```

（4）LL/SC 需要 n 个地址才能实现公平。CAS 只需要两个地址。最主要的是保证每个进程希

望获得锁的信息不会被覆盖掉。CAS 可以保证，而 LL/SC 可能会失败。

3、答：

（1）n 个共享存储处理器，每个有 m 内存。进行 O(nm)个变量排序就可能出现超线性加速比。一些随机算法也可能出现。很多搜索算法也会有超线性加速比，例如通过剪枝使得总的被搜索数据量少于顺序搜索的情况，此时并行执行将完成较少的工作。

（2）Amdahl 定律定义是：使用某种较快执行方式获得的性能提高，受到可受这种较快执行方式的时间所占比例的限制。Speedup = 1 / ((1-p)+k*p)。加速因子 k 可以比 1/n 增长快，即所谓超线性。因此并不矛盾。

4、答：

避免多个处理器同时写的不同变量处于同一 cache 行上。

5、答：

（1）可能 P1 已经把 A 替换出去了，随后 P1 又希望访问 A，因此向目录发出了 A 的 miss 请求。如果网络不能保证 P1 发出的替换 A 消息和 miss 访问 A 消息达到目录的顺序，后发出的 A 的 miss 请求越过先发出的 A 的替换请求，先到达目录，就会产生上述现象。

（2）两种可行的处理方式：a)网络保证点到点消息的顺序性；b)目录发现不一致时，暂缓 miss 请求的处理，等待替换消息到达后，目录状态正确后，再返回 miss 请求的响应。

6、答：

| 事件 | A 状态 | B 状态 |
|---|---|---|
| 初始 | I | I |
| CPU A 读 | S | I |
| CPU A 写 | M | I |
| CPU B 写 | I | M |
| CPU A 读 | S | S |

7、答：

（1）P1 对 A 的写被网络阻塞，一直没有传播到 P2 和 P3。

（2）无须施加限制。

（3）

```
P1                  P2                  P3
                    barrier1            barrier1
                                        barrier2
A=2000              while (B!=1) {;}    while (C!=1) {;}
B=1                 C=1;                D=A
barrier1            barrier2
barrier2
```

8、答：

（1）顺序一致性：

i. a=1 先于 print a 执行，b=1 先于 print b 执行。最终结果：a=1, b=1

ii. a=1 先于 print a 执行，print b 先于 b=1 执行。最终结果：a=1, b=0

iii. print a 先于 a=1 执行，b=1 先于 print b 执行。最终结果：a=0, b=1

（2）弱一致性：

除顺序一致性中的三种执行序外，还可以有第 iv 中执行序：

print a 先于 a=1 执行，print b 先于 b=1 执行。最终结果：a=0, b=0