1.

（1）Define and compare SPMD and SIMD

SPMD表示单程序多数据流模型，是一种编程模型，它允许多个处理器在同一程序控制下并行执行相同的任务，但可能在不同的数据上进行操作。

SIMD表示单指令多数据流指令集，是一种体系结构，它允许单个指令同时处理多个数据元素，这些数据元素通常存储在向量寄存器中。

SPMD与SIMD都是可以处理多数据的，不同点是SPMD是从程序级上看的，这意味着处理的多数据不一定是执行相同的操作，因为程序里面可以有分支等，即执行路径可以是多条。SIMD是从指令级上看的，这意味着SIMD处理的多数据是执行相同的操作。应用上的不同为：SPMD适用于需要大量计算的任务，而SIMD适用于需要对大量数据执行相同操作的任务。

（2）Do parallel algorithm implementations always run faster than serial algorithm implementations when solving the same problem? Explain.

不是。

因为并行化需要额外的开销，比如线程通信，线程创建，切换，销毁等。对于算法简单，问题规模较小的情况下串行可能更快。如果算法每一步直接具有依赖性，此时也不适合并行化。并行化还受到硬件的限制，如果CPU核数较少也可能会导致串行化性能优于并行化。

（3）What is the maximum independent set size that can be found in a given linked list? Explain.

链表的元素数量为n，那么最大独立集大小为n/2。

在单链表中选择尽可能多的节点，并且要保证节点之间不相邻，可以每选取第奇数个节点或者第偶数个节点，因此最大独立集的大小是链表长度的一半。

（4）Compare the array scan operation and the list ranking operation.

- 数组扫描是一种并行前缀求和操作，通常用于数组上。列表排序是一种并行操作，用于计算链表中每个节点的排名。
- 数组排序是给定一个数组 A和一个二元运算符，数组扫描的结果是一个新数组 S，其中每个元素 S[i] 是 A[0] 到 A[i] 的累积结果。列表排序的结果是一个数组 ，其中第i个元素是链表中第 i 个节点的排名。
- 数组扫描的串行实现的时间复杂度为O(n)，并行实现时间复杂度为O(logn)。链表排序的串行实现的时间复杂度为 O(n)。并行实现的时间复杂度为O(log n)。
- 数组扫描用于计算前缀和或其他前缀聚合操作。列表排序用于计算链表中节点的排名或距离。

2.

（1）When should we use OpenACC instead of OpenMP?

当需要在GPU上执行并行计算时，还有当要处理的算法需要在大型数据集上执行大量的并行计算时，使用OpenACC可以提供比OpenMP更高的性能加速。

（2）What is the problem of randomly selecting a loop to parallel? How to solve the problem?

问题：可能会选择一个并不适合并行的循环，比如循环中存在对共享资源的访问，线程之间可能会产生竞争，导致性能下降。随机选择的循环可能不是程序耗时的瓶颈。对于迭代次数较少的循环，启动并行线程的开销可能比循环本身执行的时间还要长。

解决方法：使用性能分析工具来分析每个循环，找到耗时的循环，针对性的对这些循环并行化

（3）Why does profile-driven programming have to be an iterative procedure instead of a pipeline without iterations?

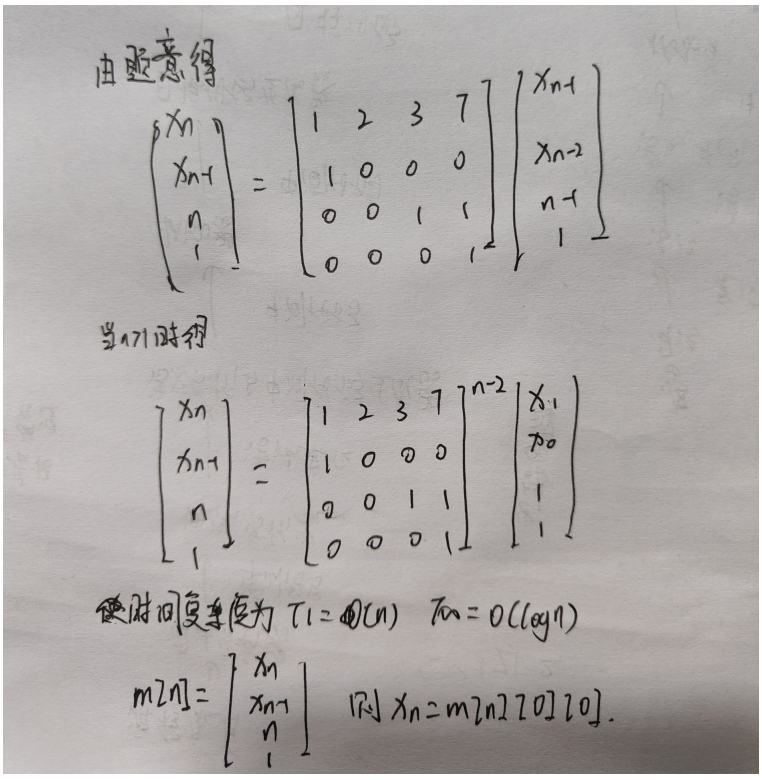
profile-driven programming是一个持续的迭代过程。因为需求会不断地变化，会导致新的性能瓶颈的出现。性能分析工具也无法一次性捕获所有性能问题。除此之外在资源有限的情况下，一次性解决所有的性能问题是不可能的。因此这是一个持续的迭代过程，需要不断的进行优化和修改。

（4）How does data movement and loop mapping help to achieve faster GPU programs?

data movement减少了数据在CPU和GPU之间的传输。将频繁访问的数据移动到共享内存中，可以显著提高内存访问效率。

loop mapping可以最大化GPU的计算资源利用率，减少线程闲置和同步开销，从而提高程序的并行效率和整体性能。

3.




```
#pragma omp parallel for
for i from 2 to N:
    x[i]=m[i][0][0]
```

算法的时间复杂度为$T1 = O(n), T\infty = O(logn)$

4.

将数组池看做一个图，使用并查集算法找到里边的所有的连通分量，并记录下连通分量的数量m。在遍历过程中，记录下该数组池的总出度k。n-k就代表了出度为零的节点数量。在一个连通分量如果存在出度为 0 的节点，那么就无法成环，并且每个连通分量中，出度为 0 的点最多只能有一个。所以环的数量为：m-n+k

```
int find(int x) {
    while (f[x] != x) {
        x = f[x];
    }
    return x;
}

bool unite(int x, int y) {
    int fx = find(x);
    int fy = find(y);

    if (fx == fy) {
        return false;
    } else {
        if (fx < fy) {
            f[fx] = fy;
        } else {
            f[fy] = fx;
        }
        return true;
    }
}
int main(){
    for(i=1;i<=n;i++)
    {
        if(next[i] = NULL)
        {
            k++;
            if(unite(i,next[i]))
            {
                m--;
            }
        }
    }
    int cnt= m-(n-k);
}
```

设每个连通分量的最大深度为m，则find操作的时间复杂度为O(m)，unite操作的时间复杂度为O(m)，整体时间复杂度就为$T1 = O(n), T\infty = O(1)$

6.

学到了多线程编程思想，如何通过并行的算法思想优化程序。最重要的一点就是合理的利用自己的硬件条件，改进自己的代码，通过增加线程更加充分的利用CPU资源。