lab3

PB21051012 刘祥辉

算法设计

1.串行代码

```
K_means():
    read N*M-dimension datas
    randomly generate K points as centers
    while iteration < max:
        for data in datas:
            for center in cluster centers:
                calculate distance of data and center
            determine center that data belongs to
        for center in cluster centers:
            calculate new coordinate of center
```

1.1 问题分析

可并行化的部分：

- 计算数据点与聚类中心的距离：在每个数据点上的距离计算可以并行进行，因为每个数据点的计算是独立的。
- 确定数据点所属的聚类中心：对于每个数据点来说，确定其所属的聚类中心是独立的，因此可以并行进行。

不可并行化的部分：

- 计算新的聚类中心坐标：在每次迭代中，需要计算新的聚类中心坐标，这涉及到所有数据点的累积信息，因此不易并行化。

可能产生空等的地方：

- 在确定数据点所属的聚类中心时，如果初始的随机选择的聚类中心导致某个聚类中心不包含任何数据点，可能会导致空等待。

负载均衡问题：

- 计算数据点与聚类中心的距离是可以并行化的，但是需要确保每个计算节点的数据点分布相对均衡，否则会导致某些节点负载过重。因此，在数据分发上需要考虑负载均衡。

额外的并行化开销：

- 并行化计算距离和确定数据点所属的聚类中心可能需要额外的开销，如线程管理、数据分发和聚合结果等。此外，需要确保并行化不会引入竞态条件或数据一致性问题。

1.2 算法描述

PCA框架

Problem (问题):

- 聚类分析是一种将数据点划分为相似组的方法。K均值聚类算法旨在将数据点分成K个不同的簇，使得簇内的数据点尽可能相似，而不同簇之间的数据点尽可能不同。

  Communication (通信):

- 使用MPI实现并行计算。通过MPI_Bcast在进程间广播均值向量，MPI_Scatter将数据分发到各个进程，MPI_Allreduce用于全局计数和求和，MPI_Finalize结束MPI环境。

  Agglomeration (聚合):

- 在每次迭代中，各个进程计算局部的簇分配和局部聚类中心的更新。通过MPI_Allreduce将局部结果聚合为全局结果，以进行下一轮迭代。

  Mapping (映射):

- 算法中的每个数据点被分配到一个进程中处理，各个进程独立计算局部结果。每个进程计算部分数据点的簇分配和部分聚类中心的更新。

伪代码

```
函数 Parallel_K_means():
    读取输入数据
    初始化聚类中心

    并行执行以下步骤，直到满足停止条件：
        广播聚类中心到所有进程
        将数据分配到各个进程：
            对于每个进程：
                计算本地数据点与所有聚类中心的距离
                将本地数据点分配给最近的聚类中心
        在所有进程上执行全局聚合操作：
            合并每个进程中的局部聚类中心和数据点数量
            在所有进程上计算全局聚类中心的新位置

    返回最终的聚类结果
```

2.实验结果

强可扩放性分析，即在固定问题规模的情况下（OJ 平台），分析程序在不同核数下的性能表现。分析未能达到线性加速的可能因素。如果出现超线性加速，分析可能原因。

核数	时间/ms
1	6274ms
2	5748ms
4	4985ms
6	4325ms
8	4565ms

随着核数增加，程序的运行时间呈现了一定程度的下降，但并没有达到线性加速。这可能是由于以下几个因素造成的：

通信开销： 在多核系统中，不同核之间需要进行通信以共享数据或协调任务。随着核数增加，通信开销可能会增加，从而影响性能表现。

负载不平衡： 如果程序无法有效地分配工作负载到每个核心上，就会导致某些核心的负载过重，而其他核心处于空闲状态。这可能会导致性能的下降，因为某些核心的利用率较低。

缓存效率： 多核系统中，不同核之间共享缓存。随着核数增加，缓存的竞争可能会增加，导致缓存效率下降，从而影响程序的性能。

任务并行度限制： 在某些情况下，程序可能存在任务之间的依赖关系或者无法被有效地并行化，从而限制了程序的并行度，进而影响了性能提升的幅度。

3.总结

选好合适的算法进行并行化，不要只是并行已有的串行代码