基于CUDA编程实现的分批并行K-means算法

21307140069 田沐钊

1 简介

K-means 算法是一种应用极其广泛的聚类算法，其在执行过程中需要对每一个数据进行多次计算。假设总数据量是 N，聚类数目是 k，如果算法对全部数据进行M 次遍历之后数据的聚类情况才趋于收敛，则对每个点至少需要进行$k \times M$次计算，整个算法的时间复杂度为 $O(kMN)$。这对计算机的算力和数据吞吐量要求是较高的。如何在同等情况下，减少对数据的遍历次数，是该算法所面临的一个挑战。

本项目通过多方的资料查询，通过CUDA编程实现了一种分批的基于GPU 的K-means 算法，大大加快了算法的运行效率。

2 实现思路

2.1 相关工作

关于 Kmeans 算法的优化有许多先关的研究，比如 *Reza R, Daniel R, Ellick C, et al. A Parallel Implementation of k-Means Clustering on GPUs[C]//Proc. of International Conference on Parallel and Distributed Processing Techniques and Applications. [S. l.]: Springer-Verlag, 2008: 340-345.* 这篇工作中，其使用了带缓存机制的常数存储器保存中心点数据，能获得更好的读取效率，但这种方法仅针对1维数据，在实际应用中存在较大的局限性。

而在 *Mario Z, Michael G. Accelerating K-means on the Graphics Processor via CUDA[C]//Proc. of the 1st International Conference on Intensive Applications and Services. [S. l.]: IEEE Press, 2009:7-15.* 这篇工作中，其将同一中心点的数据都读进同一个 block 中的shared memory 中，由单个thread进行读取。但是这样每个thread同时只会计算与同一中心点的距离。这种做法依然会导致系统反复向全局存储器读取数据点的数据，所以效果也有很大的提升空间。

Bai Hongtao, He Lili, Ouyang Dantong, et al. K-means on Commodity GPUs with CUDA[C]//Proc. of WRI World Congress on Computer Science and Information Engineering, [S. l.]: ACM Press, 2009: 651-655. 这篇工作中，其将算法分成2个部分各自实现，即聚类和求中心点。在完成一次聚类操作之后，需要将聚类结果从显存拷贝回内存，统计每个类别的数据点信息，再重新传给显存，交由GPU计算新的中心点。

本项目将改善后的方法应用在BG K-means 中，对数据采取分批原则，更合理地运用CUDA 提供的存储器，如共享存储器、全局存储器、常量存储器，避免访问冲突，同时减少对数据集的访问次数，以提高算法效率。

2.2 原始 K-means 的算法描述

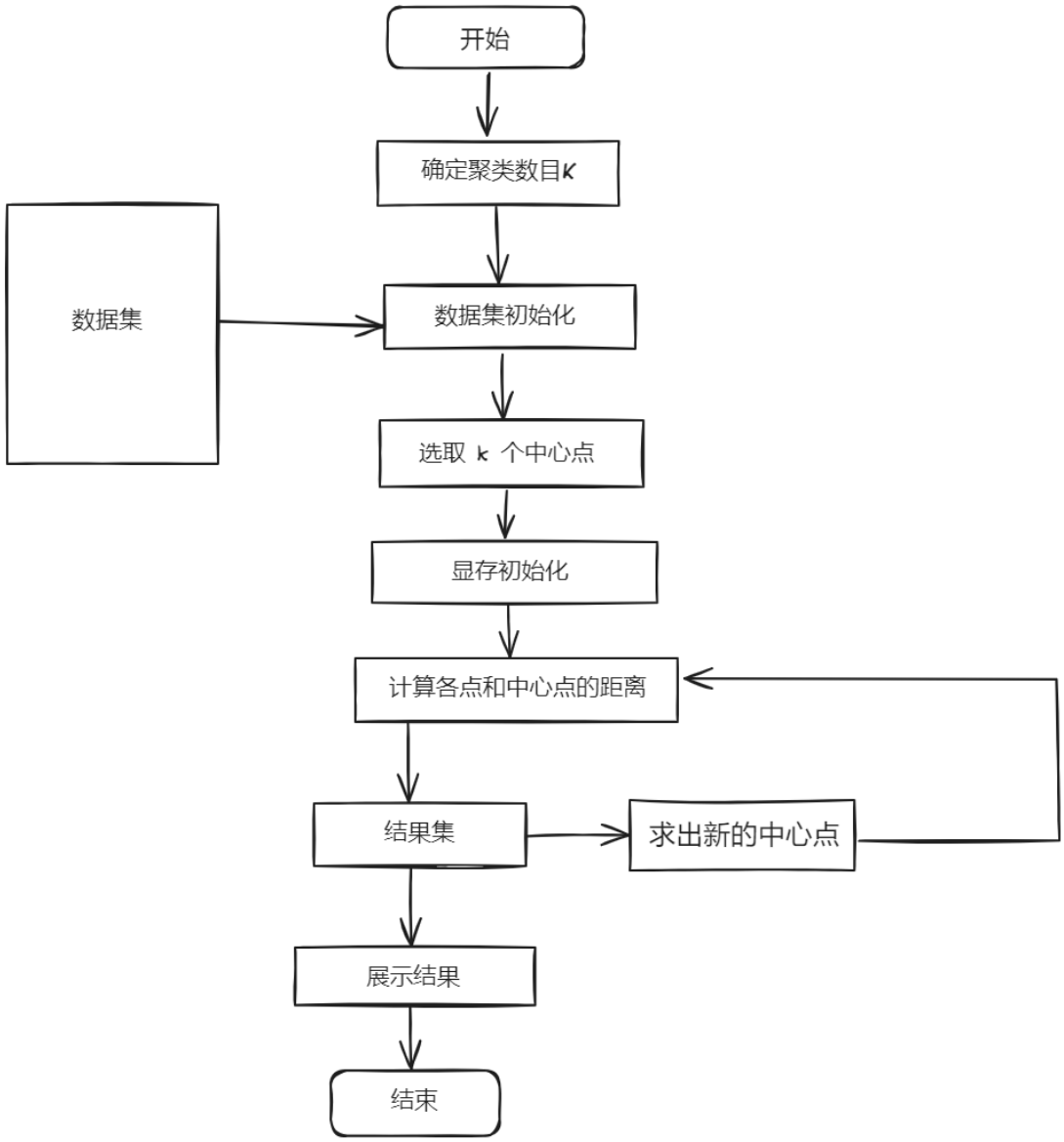
Kmeans算法的目的是通过聚类将给定的 n 个对象划分为 k 个簇，使簇内的对象相似度较高，簇间相似度较低。其具体算法流程如下：

1. 初始化 k 个聚类中心(centroids)。通常是随机选择 k 个数据点作为初始中心点。
2. 将每个数据点分配到最近的聚类中心。通常使用欧几里德距离来计算数据点到各聚类中心的距离,并将数据点划分到距离最小的聚类中心。

3. 计算每个聚类的新中心点。对于每个聚类,计算所有数据点的平均值作为新的聚类中心。
4. 重复步骤2和3,直到聚类中心不再发生变化或达到最大迭代次数。
5. 输出最终的 k 个聚类以及每个数据点所属的聚类。

2.3 并行 K-means 的算法流程

本项目实现的并行K-means 算法流程如下图所示：



算法的具体运行步骤如下：

1. 确定聚类数目k 的值。
2. 读取数据集至主机内存。
3. 随机选择 k 个点(两两互不相同)，作为每一个簇的初始中心点。
4. 初始化
    - 根据运行环境，配置线程块数及线程数。
    - 分配结果集对应的内存及显存地址空间。

- 向显存分配并拷贝必要数据，如数据集、k 个初始中心点。

5. 对数据集进行聚类
   - 由GPU 分别计算与不同中心点的距离的平方误差和。
   - 对数据集中每一个点，求出平方误差和最小的中心点所对应的分类(这里由数组小标决定)，获得聚类结果。
   - 返回isChanged 数组至内存：若算法收敛，则退出；否则，算法继续。
6. 计算新中心点
   - 在GPU 中，扫描数据集的聚类结果，累加得出每一个分类的每一个维度的数值总和，以及每一个分类的总数。
   - 在CPU 中，根据每一个分类的每一个维度的数值总和以及总数，求出每一个分类的新中心点。
   - 根据新的k 个中心点，重复步骤(5)、步骤(6)。
   - 显示结果，清算运算中分配的空间，完成运算过程。

3 具体的代码实现

3.1 变量说明

下面是程序中使用的数据结构和参数说明：

(1)COUNT：参与计算的数据对象的个数。

(2)k_COUNT：簇的数量。

(3)DIMENSION：参与计算的维度的数量。

(4)num_threads：每一个线程块中线程的数量。

(5)num_blocks：线程块的数量。

(6)中心点：保存中心点数据。一共DIMENSION 行，k_COUNT 列。在设备端，该数组被定义在常数存储器中。

(7)数据集：存放参与聚类分析的对象的全部维度数据。在形式上有DIMENSION 行COUNT 列；列下标对应每一个
对象的ID。假设：

$$width = num\_threads \times num\_blocks$$
$$row\_count = COUNT / width$$

数据集将被折成多份，组成一个row_count×DIMENSION 行、width 列的数组。

(8)结果集：聚类结果的数组，row_count 行，width 列。

(9)sumArray：用于记录每一个分类的每一个维度的数值总和，及每一个分类的总数。
$k_COUNT \times (DIMENSION + 1)$行，width 列。sumArray 的每一列的意义都是一样的。

(10)sharedSumArray：平方误差和缓存，保存在shared memory 中。分为num_threads 列，每一列由列下标对应的 thread 操作，避免bank conflict；k_COUNT 行(或kMax 行)。

(11)isChanged：用来标记聚类结果是否有变化，以表示算法是否收敛。大小为width。每一个线程负责isChanged数组的一个元素。

其中，数据集、结果集、sumArray、isChanged 都需要在主机内存以及设备显存分别定义相应的数组。在本文算法中， 保存在全局存储器的数组均定义为width 列；一共有width 个线程，每一个线程负责一列数据。

3.2 模块解释

该算法由3 个部分组成：数据仓库读取及数据初始化模块，数据聚类模块，中心点获取模块。其中，数据聚类模块主要由GPU完成，而中心点获取模块则由GPU与CPU共同完成。

3.2.1 数据聚类模块

对于 k_count 个中心点，采取分批原则。假设：

$$kMax = \frac{sharedMemPerBlock - 1}{sizeof(float) \times num\_threads}$$

$$m = K\_COUNT / kMax$$

其中sharedMemPerBlock=16 384 Byte，为每一个block 的 shared memory 的大小。则k_COUNT 次运算被分为m 批，每批累加数据集与 kMax 个中心点的平方误差和。本模块的伪代码如下：

```
FOR n = tid_in_grid TO COUNT
STEP + blockDim.x * gridDim.x DO //Loop1
(
    //取数据集中对象
    FOR k = 0 TO m DO //Loop2
    (
        FOR i = 0 TO DIMENSION DO //Loop3
        (
            //对每一个分类累加平方误差和
            currentData = g_idata[DIMENSION*n + i][ tid_in_grid] //Read Once, Use
K_COUNT Times
            FOR j = k*kMax TO (k+1)*kMax DO //Loop4
            (
                //累加结果保存在共享存储器中
                sharedSumArray[blockDim.x * (j-k*kMax) +threadIdx.x] += square (
currentData – de_k_Points[i][j] )
            ) //End Loop4
        ) //End Loop3
        FOR i = 0 TO kMax DO //Loop5
        (
            //找出最小平方误差和
            min(sharedSumArray[blockDim.x * i + threadIdx.x] )
        ) //End Loop5
    ) //End Loop2
) //End Loop1
```

在算法1 中Loop1 将由主机端执行。执行完Loop1 后，将isChanged 数组返回CPU，判断算法是否收敛。

当num_threads=256 时，kMax=15；num_threads=128 时，kMax=31。

当k_COUNT≤15 或k_COUNT >31 时，可使用算法1。此时num_threads=256，kMax=15。当15 < k_COUNT≤31 时，则可以取num_threads=128，kMax=31。

3.2.2 中心点获取模块

该模块需要由GPU 和CPU 共同完成。

GPU 部分主要仍是一个kernel 函数，用于生成sumArray数组。此函数定义线程仅负责一列数据的统计。在完成创建sumArray 的过程后，将其回传至内存中，对于每一行，由CPU 完成num_threads×num_blocks 次累加，总共产生k_COUNT×(DIMENSION+1)个数据，也就是每一个分类的每一个维度的数值总和，以及每一个分类的总数，然后就能求出相应维度的平均值，也就是新中心点的值，以结束该模块的运算。

4 算法测试

本文基于color数据集进行测试，该数据集中包含1000000个9维向量。

该并行算法在该数据上不同聚类参数的耗时结果以及与串行算法耗时的比较如下：

聚类数	**4**	**8**	**16**
并行耗时(s)	22.70	21.779	21.691
串行耗时(s)	902.37	901.66	898.49

可见该算法的性能提升显著。

并且经测试，在结果上与串行算法的准确度并无明显差异。

5 总结

本文提出了一套基于CUDA 的K-means 算法，以及相应的数据结构。算法采用分批原则进行聚类计算，最大限度地运用了CUDA 中最快的2 种存储器，并减少了对慢速存储器的存取。
但是，该K-means算法的收敛条件定义得比较严格，收敛速度慢，并且CUDA 提供的计算精度有限。今后的研究将致力于加快算法收敛速度，以及提高运算精度。