第38卷 第13期 计 算 机 工 程 2012年7月
Vol.38 No.13 Computer Engineering July 2012

·人工智能及识别技术· 文章编号：1000—3428(2012)13—0145—03 文献标识码：A 中图分类号：TP301.6

分批处理的 K-means 算法并行实现

兰远东，刘宇芳，徐 涛

(惠州学院计算机科学系，广东 惠州 516007)

摘 要：为解决 K-means 算法计算量大、收敛缓慢、运算耗时长等问题，给出一种新的 K-means 算法的并行实现方法。在通用计算图形处理器架构上，使用统一计算设备架构(CUDA)加速 K-means 算法。采用分批原则，更合理地运用 CUDA 提供的各种存储器，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。在大规模数据集中的实验结果表明，该算法具有较快的聚类速度。

关键词：数据挖掘；K-means 算法；统一计算设备架构；并行算法；聚类分析；图形处理器

Parallel Implementation of K-means Algorithm with Batch Processing

LAN Yuan-dong, LIU Yu-fang, XU Tao

(Department of Computer Science, Huizhou University, Huizhou 516007, China)

【Abstract】K-means algorithm is computationally intensive, time consuming and convergence slow. In order to solve the problem of K-means algorithm, a new set of parallel solution of K-means algorithm is presented. In the General Purpose computation on Graphics Processing Unit(GPGPU) architecture, Compute Unified Device Architecture(CUDA) is used to accelerate K-means algorithm. Based on batch principle, the algorithm uses CUDA's memory more rationally, to avoid access conflict, reduce the number of times of visits for data sets, and improve the efficiency of K-means algorithm. Experimental result in large-scale data set shows that the algorithm has a faster clustering speed.

【Key words】data mining; K-means algorithm; Compute Unified Device Architecture(CUDA); parallel algorithm; clustering analysis; Graphics Processing Unit(GPU)

DOI: 10.3969/j.issn.1000-3428.2012.13.043

1 概述

K-means 算法[1]是聚类分析算法中，最主要的算法之一，也是应用得最广泛的算法之一[2]。然而 K-means 算法的一大特点就是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，当算法对全部数据进行 M 次遍历之后，算法收敛，即数据的聚类情况不再变化，则对每个点至少需要进行 $k\times M$ 次计算，整个算法至少需要 $k\times M\times N$ 次计算。当 N 比较大，整个算法的计算量将非常大。这样的计算量，除了要求计算机拥有强大的计算能力，更要求系统对数据的吞吐量不能太低。如何克服这些缺点，将是继续发展 K-means 算法的一个关键课题。同时，如何在同等情况下，减少对数据的遍历次数，也是该算法所面临的一个挑战。

鉴于上述现状，本文结合 K-means 算法，以及统一计算设备架构(Compute Unified Device Architecture, CUDA)的编程规范和技巧，提出一套能应用在 GPU 上的 K-means 并行算法，称为分批的基于 GPU 的 K-means 算法(Batch GPU-based K-means, BG K-means)，旨在充分发挥 GPU 强大的计算能力，超高片内带宽，加速 K-means 算法。

2 相关研究

有大量将 K-means 应用于实际需求的研究，主要面向顾客类型及行为分析、产品定位等。基于 GPU 的谱聚类算法的实现[3]：将用于谱分析方法的正规矩阵应用数据集的规范化上，并将这些数据传给 K-means 算法进行分析，能得到更快、更准确的分析结果。文献[4]使用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。但该算法仅针对 1 维数据；(总数据量:中心点数)=(256:1)。这些情况在实际应用中很少出现。在文献[5]将同一中心点的数据都读进同一个 block 中的 shared memory 中，由每个 thread 读取；每个 thread 同一时间都只计算与同一中心点的距离。这种做法会导致系统反复向全局存储器读取数据点的数据。数据点的数量往往比中心点的多很多倍。文献[6]将算法分成 2 个部分，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，统计每个类别有多少个数据点，分别是哪些，再将这 3 个数据结构重新传给显存，交由 GPU 计算新的中心点。

本文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。

3 K-means 并行实现

3.1 K-means 算法描述

K-means 算法以 k 为输入参数，将给定的 n 个对象，划分为 k 个簇，其中簇内的对象相似度高，而属于不同簇的对象的相似度低[7]。

K-means 算法的流程如下：

(1)在给定的 n 个对象中，随机选取 k 个对象，作为每一个簇的初始均值(中心)。

(2)对于其余的 $n–k$ 个对象，分别计算与 k 个中心的距

基金项目：国家“863”先进制造领域基金资助重点项目(2006AA04A120)；广东高校优秀青年创新人才培养计划基金资助项目(LYM09128)

作者简介：兰远东(1975－)，男，博士研究生，主研方向：模式识别，机器学习；刘宇芳，副教授；徐 涛，博士研究生

收稿日期：2011-10-18 **E-mail：**lanyuandong@gmail.com

离，将对象指派到最接近的簇。

(3)计算每个簇的新均值，得出 k 个新的中心。

(4)根据 k 个新的中心，重复步骤(2)、步骤(3)，直至准则函数收敛。

k 个新的中心不一定属于原有的 n 个对象。

通常，采取平方误差准则计算点间距离，定义如下[8]：

$$E=\sum_{i=1}^{k}\sum_{p\in C_i}\left|\boldsymbol{p}-\boldsymbol{m}_i\right|^2 \tag{1}$$

其中，E 是数据集中所有对象的平方误差和；$\boldsymbol{p}$ 是给定的 n 个对象；$\boldsymbol{m}_i$ 是簇 C_i 的均值(即中心)；$\boldsymbol{p}$ 与 $\boldsymbol{m}_i$ 均为多维向量。

3.2 BG K-means 流程

本文给出的 BG K-means 算法流程如图 1 所示。

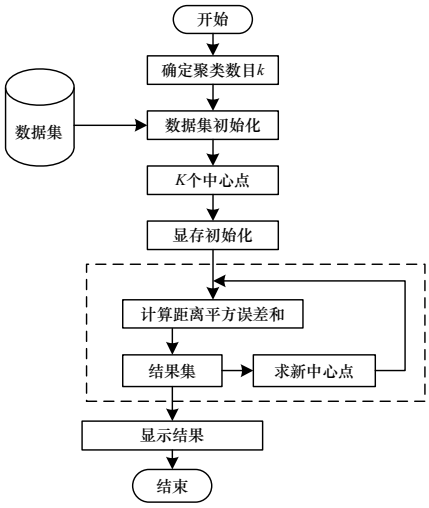


图 1 BG K-means 算法流程

根据算法流程，给出算法的具体步骤如下：

(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

1)根据运行环境，配置线程块数及线程数。

2)分配结果集对应的内存及显存地址空间。

3)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

1)由 GPU 分别计算与不同中心点的距离的平方误差和。

2)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。

3)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

1)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。

2)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。

(7)根据新的 k 个中心点，重复步骤(5)、步骤(6)。

(8)显示结果，清算运算中分配的空间，完成运算过程。

3.3 使用的数据结构

在算法实现过程中，使用到的主要数据结构和参数说明如下：

(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：

$$width=num\_threads\times num\_blocks \tag{2}$$

$$row\_count=\lceil COUNT/width\rceil \tag{3}$$

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。共 *k_COUNT*×(*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*[1]；*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*(2)。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。

3.4 算法实现

本文算法主要由 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。

3.4.1 数据聚类模块

对于 *k_COUNT* 个中心点，采取分批原则。假设：

$$kMax=\left\lfloor\frac{sharedMemPerBlock-1}{sizeof(float)\times num\_threads}\right\rfloor \tag{4}$$

$$m=\lceil K\_COUNT/kMax\rceil \tag{5}$$

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：

算法 数据聚类

```
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，有 2 个办法：取 *num_threads*=128，*kMax*=31(方法 1)，或者取 *num_threads*=256，*kMax*=15(方法 2)。对于不同的 GPU，存在不同的临界点，假设为 *CRITICALk_COUNT*。当 *k_COUNT*< *CRITICALk_COUNT* 时，方法 1 有优势；当 *k_COUNT*≥*CRITICALk_COUNT* 时，方法 2 有优势。这主要受 GPU 线程执行效率和显存带宽的影响。

3.4.2　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。图 2 是一个生成 *sumArray* 的例子。

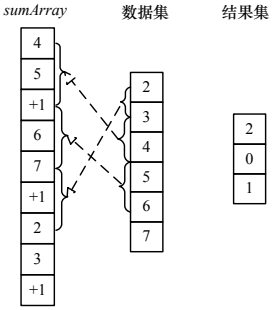


图 2　*sumArray* 生成举例

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　算法性能测试

4.1　测试环境

测试环境如下：GPU 为 GeForce 9600 GT，1 GB 显存；CPU 为 AMD Athlon(tm) 64 X2 Dual Core Processor 5200+，8 GB 内存；操作系统为 Ubuntu 8.10 64 bit。开发基于 CUDA 2.3 进行。测试数据的 Dimension 为 50。

4.2　数据仓库

本文采用 kDD-CUP-98 数据源，kDD-CUP-98 的数据来源于 Paralyzed Veterans of America(PVA)。PVA 希望通过工具，分析最近一次资金筹措活动的结果。

本文基于 kDD-CUP-98 构建数据仓库，基于星型模型构建，包含一个事实表和一个维度表。

捐赠者行为事实表：ID，年龄，是否为房主，子女数，家庭收入，性别，购买各种书籍的次数，邻居及生活小区情况，如小区房价、人种比例。这些数据都需要进行最值规范化处理。

ID 维表：ID 和会员 ID 的对应关系。

在事实表中，对于区间变量，采取最值规范化；对于序数变量，采取以下方式进行规范化：假设变量 *A* 共有 *C* 个序数，分别是$\{v_1,v_2,\cdots,v_c\}$，则先用$\{1,2,\cdots,C\}$来代替相应数据，再根据以下公式进行预处理：假设 *o* 为原数值，*o*’为规范化后的值，则：

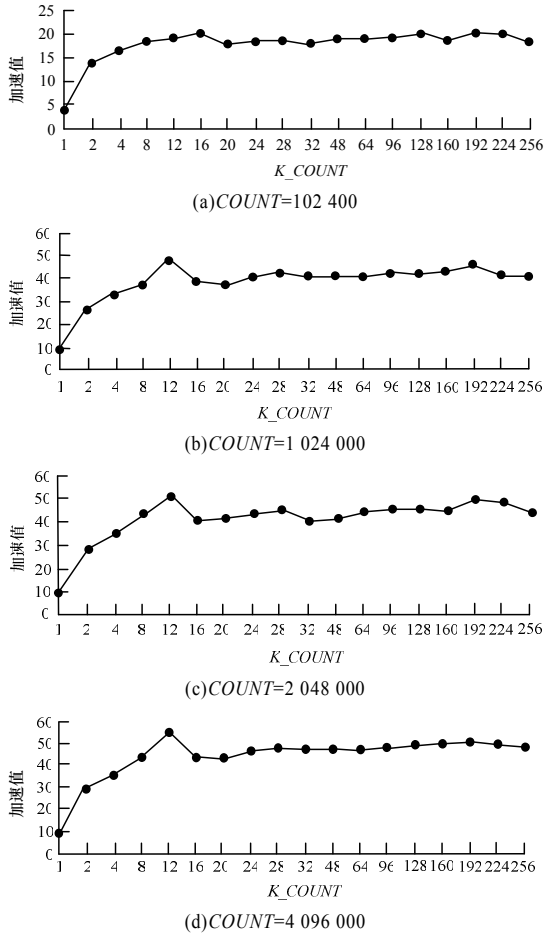
$$o' = \frac{o-1}{C-1} \tag{6}$$

为减小数据仓库的大小，可以清除购买次数几乎为 0 的数据，或者进行数据合并。

算法所用到的 ID 并不一定对应一个指定的客户或商品，也可以指代一批相似的客户或一类商品。因此，可以在保持原数据比例的前提下，合并相似项。若过度合并，则该数据很可能会被淹没在其余数据中。

4.3　聚类结果

运行条件如下：*COUNT*=4 163 850, *k_COUNT*=15，*DIMENSION*=52, *num_blocks*=24, *num_threads*=256。运算过程平均耗时约 42 s，循环大约 100 次之后算法收敛；相对而言，CPU 运算则需要约 31 min，GPU 提速 45 倍，测试性能如图 3 所示。



(a)*COUNT*=102 400

(b)*COUNT*=1 024 000

(c)*COUNT*=2 048 000

(d)*COUNT*=4 096 000

图 3　测试性能

有 4 个主要消费群：(1)男性公民，年龄偏大，屋主，家庭收入中等水平；社区收入中等水平。(2)男性公民，年龄偏大，屋主，家庭收入中等偏上；所在社区生活水平较高，黑

(下转第 151 页)