文章编号:1671-4598(2018)02-0273-03 DOI:10.16526/j.cnki.11—4762/tp.2018.02.067 中图分类号:TP338.6 文献标识码:A

基于滑动门中心点计算的K均值聚类并行算法研究

龚运鸿，周新志，雷印杰
（四川大学 电子信息学院，成都 610065）

摘要：随着GPU硬件设备的普及和GPGPU技术的快速发展，越来越多的研究人员投入到GPGPU的研究当中；当前，GPU具有很强大的并行计算能力、浮点运算能力、计算单元集成能力等特点，显示出了GPU在并行计算领域的巨大潜力；CUDA是由NVIDIA公司提出的一种利用GPU进行并行计算的架构，CUDA使得GPU具有友好的可编程性，为研究人员能够在GPU上实现各种领域的科学计算提供了方便的途径；K均值聚类算法由于其概念简单，易于实现等优点成为并行计算研究的一个热门方向；对于K均值并行算法的研究，有基于8核CPU并配备FPGA加速板的方法，但对于一个需要启动数千个线程的复杂模型，基于传统CPU并行计算方法难以实现；也有使用CUDA并行计算平台对K均值聚类算法进行处理，但处理算法时通常忽略对CUDA平台上K均值聚类算法自身的优化；基于以上缺陷，介绍K均值聚类算法的同时对算法在CUDA平台上进行了相应优化，特别针对更新中心点的耗时问题，提出了一种基于滑动门中心点计算的K均值聚类并行计算；实验结果表明，当聚类数较多时，相对于传统的更新中心点算法，基于滑动门中心点并行算法的效率更高。

关键词：K均值；并行；CUDA；GPU；滑动门

Research on K—means Clustering Parallel Algorithm Based on Sliding Gate Center Point Calculation

Gong Yunhong, Zhou Xinzhi, Lei Yinjie
(College of Electronics and Information Engineering, Sichuan University, Chengdu 610065, China)

Abstract: With the popularity of GPU hardware and the rapid development of GPGPU technology, more researchers have invested in GPGPU research. Because of the strong parallel computing power, floating—point computing power, computing unit integration capabilities and other characteristics, GPU shows the great potential in the field of parallel computing. CUDA is an architecture developed by NVIDIA for parallel computing using a GPU and it makes the GPU highly user—friendly and provides a convenient way for researchers to implement scientific computing in a variety of fields on the GPU. K—means clustering algorithm has become a popular direction for parallel computing because of its simple concept and easy realization. For the K—means parallel algorithm, there is a 8—core CPU and FPGA—based accelerator board method, but for a complex model that needs to start thousands of threads and the traditional CPU parallel computing method is difficult to achieve; What's more, some researches have study the K—means clustering algorithm based on the CUDA parallel computing platform, but the studies usually ignore the algorithm optimization. Based on the above shortcomings, the K—means clustering algorithm is introduced on the CUDA platform, and the K—means clustering parallel computation based on sliding gate center point calculation is proposed. The experimental results show that when the number of clusters is large, the parallel algorithm based on the sliding gate is more efficient than the traditional updating center algorithm.

Keywords: k—means; parallel; CUDA; GPU; sliding gate

0 引言

随着程序需要处理的数据量越来越庞大，现如今以GB或TB为单位的数据集已经十分普遍了，数据挖掘中必须重视的一个问题就是如何高效得处理如此庞大的数据。即使算法的复杂程度是线性增长的，时间和空间的消耗也不容忽视。K均值聚类算法由Stuart Lloyd等人在1957年第一次提出[1]，K均值聚类算法源于信号处理的一种矢量量化方法，由于其概念简单、收敛速度快、易于实现等特点，现如今在数据挖掘领域聚类分析中十分流行。然而K均值聚类算法的复杂度比较高，如何高效进行算法计算是一个急切的研究方向。目前，陶冶[2]等人证明并实现了并行K均值聚类算法。喻金平[3]等人提出一种改进的人工蜂群算法，解决了K均值算法的搜索能力差的问题。霍莹秋[4]等人提出分块、并行的K均值聚算法，采用“合并访问”、“多级规约求和”和“负载均衡”等优化策略优化并行算法，提高了算法的运行速度。对于K均值聚类并行计算的研究还存在许多缺陷，比如没有针对CUDA并行计算平台进行优化，也没有针对中心点更新效率问题提出解释等。根据以上研究的缺陷，本文利用NVIDIA的CUDA并行平台，在传统K均值算法基础上，采用了一种滑动门并行计算中心点算法优化K均值算法更新中心点的耗时问题。通过与规约法计算中心点算法相比，获得了很好的加速比。

1 CUDA并行计算平台

随着社会的发展，CPU逐渐达到了速度极限并且购置成本也在急速上升。通过对显示图像的优化，GPU技术却逐渐

收稿日期:2017-08-18； **修回日期**:2017-08-31。

作者简介:龚运鸿(1990-),男,江苏无锡人,硕士研究生,主要从事智能控制方向的研究。
周新志(1966-),男,四川德阳人,教授,硕士研究生导师,主要从事智能控制方向的研究。

完善了起来，在通用计算领域，GPU 的能力逐渐超过了传统 CPU。通过近几年的发展，计算技术正在从单一 CPU 串行计算方式向 GPGPU 并行协同计算发展。为了让开发者无需学习复杂的着色语言和图像处理原语，方便更多的开发者能够进行 GPU 编程，显卡厂家 NVIDIA 在 2007 年提供了一个方便开发者使用的接口——Compute Unified Device Architecture，即 CUDA。CUDA 可以让开发者直接访问 GPU 的虚拟指令设定和并行计算元素。与在 GPU 上使用图形 API 进行计算的传统方式相比，CUDA 具有十分巨大的优势：1）程序可以在内存的任意位置进行读取；2）在 CUDA4.0 以上的版本拥有统一虚拟内存；3）在 CUDA6.0 以上的版本拥有统一内存；4）CUDA 为开发者开辟一个快速共享内存区域，其数据可以在线程中共享。这可以帮助开发者管理缓存、建立更高的带宽；5）可以更快地从 GPU 上下载和回写数据。在科技研究方面，CUDA 技术可谓炙手可热，越来越多的研发人员投入到了 CUDA 并行技术研究当中，所以在科研领域，CUDA 也拥有了许多研究成果：在医疗领域，可以将 CUDA 加速技术运用在 CT 或者 MRI 扫描图像的 VR 技术上；在物理建模上，特别是流体动力学领域有很好的成果；在神经网络训练领域中，CUDA 技术可以很好地解决机器学习遇到的瓶颈。

CUDA 将 C 语言进行扩展，这样可以使开发者使用 C 语言等高级语言在 GPU 设备上进行并行程序编程，方便开发者使用。使用 CUDA 进行编写的代码，既可以在 CPU 上使用，也可以在 GPU 上使用。使用 CUDA 并行计算平台时，主机端为 CPU，设备端为 GPU。在 GPU 运行时，基于 NVIDIA 自身底层硬件特点，采用了单程序多数据（SPMD）的并行计算方式来处理数据，属于单指令多数据（SIMD）的一种变体。并行编程的核心是线程的概念，运行的程序称为内核函数（kernel），并行计算时，每一个线程都会同时处理一个 kernel，CUDA 中数据通过线程－块－网格计算方式进行分配。每个线程块都有自己唯一的识别 id，一个或者多个线程块会由流多处理器 SM 来并行处理，每一个 SM 由很多个 32 位寄存器组成。每个 SM 中有 8 个流处理器（SP）构成。每一个流处理器都有一个共享存储器，所有 SP 都可以访问共享存储器中的内容。如果线程块是一维的，那么 blockId.x 就可以确定线程块的 id 了。如果线程块是二维的，那么需要 blockId.x 和 blockId.y 才能确定线程块的 id。每一个线程也有自己唯一识别 id，通过这个 id 查找需要处理数据的位置。如果线程是一维的，那么 threadIdx.x 就可以确定线程的 id 了。如果线程是二维的，那么需要 threadIdx.x 和 threadIdx.y 才能确定线程的 id。如果线程是三维的，那么需要 threadIdx.x，threadIdx.y，threadIdx.z 这 3 个参数才能确定线程的 id。同一线程块中的线程如果需要进行数据传输，一般是在共享存储器中进行的。

2 K 均值聚类算法

2.1 K 均值聚类算法介绍

聚类是一种无监督的学习，它将相似的对象归到同一簇中，簇内的对象越相似，聚类的效果越好。K 均值聚类算法可以发现 k 个不同的簇，且每个簇的中心采用簇中所含值的均值计算而成。

K 均值聚类算法是将含有 n 个数据点划分为 k 个簇 C_j（$j=1,2,\ldots,k;k\leqslant n$）。首先在 n 个数据点中随机选取 k 个数据点代表 k 个簇的质心，再根据剩余数据点与 k 个质心的距离，将剩余数据点划分到与其距离最近的簇 C_j 中。然后重新计算每个簇中所有值的均值，并将这个均值作为新的质心。该过程不断重复，直到误差平方和函数 SSE 收敛，其定义如公式（1）所示：

$$SSE=\sum_{j=1}^{k}\sum_{x\in C_j}x-C_j^{\,2} \tag{1}$$

$$c_j=\frac{1}{m_j}\sum_{x\in C_j}x \tag{2}$$

式中，k 代表选择的簇的数量；C_j 代表第 $j(j=1,2,\ldots,k;k\leqslant n)$ 个簇；x 代表簇 C_j 中的任意一个数据点；c_j 是簇 C_j 的均值；m_j 代表簇 C_j 中数据点的个数。

2.2 K 均值聚类流程

1）初始化聚类中心。在 n 个数据点钟随机选取 k 个数据点作为初始聚类中心 $C_1,C_2,\ldots,C_k$。

2）计算剩余数据点到每个初始聚类中心的距离，将剩余数据点几个划分到与其最近的簇中心代表的簇中，根据公式（1）计算 SSE 的值。

3）分别算出各个簇中所有数据点的均值，用这些均值替换初始聚类中心，用新的聚类中心重复步骤 2）。

4）将上一次 SSE 的值和本次 SSE 的值进行比较，差值绝对值大于阀值，代表 SSE 收敛，则进行步骤 5），否则进行步骤 5）。

5）算法结束。

排除其他因素只考虑算法核心部分，把每一次的比较、乘法、加法操作都当做一次浮点运算，用 T_{flop} 来代表一次浮点运算花费的时间，则算法核心部分每次迭代所用的时间见表 1。

表 1 K 均值聚类算法核心部分所用运算时间

步骤	运算时间
2	$(K+1)ndT_{flop}$
3	nT_{flop}

3 数据准备阶段并行设计

3.1 初始化数据的并行设计

选取初始化聚类中心的过程，不是完全随机产生的，需要先确定数据点在所有维度的最大最小值，如图 1 所示。

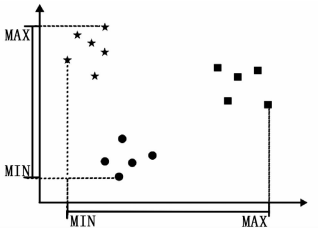

图 1 二维数据点分布图

在选取初值时，求最大最小值是典型的规约运算，而规约运算是可以并行化的。对于 n 个输入数据和操作⊕，规约可表示为：

$$\sum_1^n a_i=a_0\oplus a_1\oplus a_2\oplus\ldots\oplus a_n \tag{3}$$

图 2 展示了处理 N 个元素规约操作的实现。

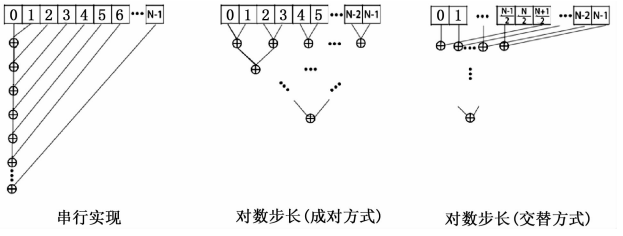

图 2　3 种不同规约操作的实现

由图 2 可以发现，不同的实现方式，其时间复杂度也是不同的。其中，串行实现完成规约操作需要 $n-1$ 步，两种对数步长的规约操作只需要 lg$n$ 步，大大减少了时间复杂度。但是，由于成对方式不能合并内存事务，成对方式在 CUDA 实现中性能比较差。在 CUDA 中，交替方式在全局内存和共享内存都有很好的效果。在全局内存中，将 blockDim. x * grid-Dim. x 的倍数作为作为交替因子，这样可以获得比较好的性能。在共享内存中，需要保持线程块相邻的线程保持活跃状态，同时，为了避免内存的冲突，需按确定的交替因子来累计结果，这样可以获得良好的效果。

3.2　分配数据的并行设计

基于 GPU 的 K 均值聚类算法在进行数据对象分配这一步骤的时候有两种策略。第一种策略是面向每个簇的中心，通过计算出该簇的中心与每个数据对象的距离，然后将每个数据对象归并到距离簇中心最近的那个簇中。这种策略适应于 GPU 的处理核心数量比较少的情况，此时，GPU 中的每个处理核心可以对应一个簇的中心，并且能够连续的处理所有的数据对象。另一种策略是面向每个数据对象，每个数据对象计算与所有簇中心的距离，然后数据对象将会被分配到距离簇中心最近的那个簇当中。该策略适应于 GPU 的处理核心比较大的情况，目前主流的 GPU 一般都有超过 100 个处理核心，因此第二种处理策略比较合适。

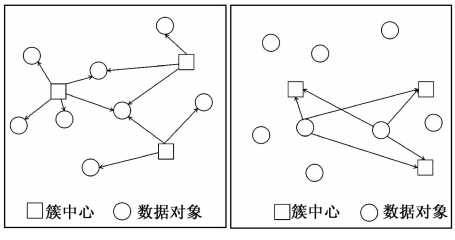

图 3　数据分配过程不同策略对比图

4　滑动门中心点并行算法设计

在目前的 CUDA 并行计算算法优化中，一般使用带状划分的并行划分方法[5-8]，如图 4 所示，K 均值聚类算法的中心点存储在共享内存中，而样本保存在全局存储器中，每一行数据由一个线程进行处理。带状划分方法能最大化地利用 GPU 的计算内核。

采用带状划方法的方法，线程 1～m 将在同一时间去读取共享储存器索引位置 0 的数据。在 CUDA 中，寄存器的访问速度最快。所以为了减少重复访问的时间，可以把共享存储器中的数据转化到寄存器当中进行存储。

采用滑动门并行算法时，也是采用带划分的划分方式，在每个块上开启 m 个线程，相同簇内的样本都会在这些线程中

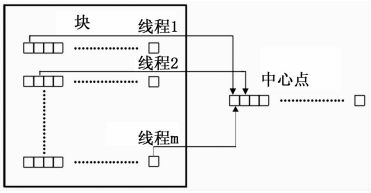

图 4　带状划分的并行聚类算法

进行计算。每个块中的样本点值最后合并至临时中心点存储区 s _ cust 中。m 的最佳值由式（4）计算。

$$m = \min\{L - L\%w_{size}C_i - C_i\%w_{size}, tNum, \frac{SM_{size}}{4}\} \quad (4)$$

式中，L 代表样本向量的维度；w_{size} 代表 wrap 的大小；C_i 代表簇 i 中的样本总数；$tNum$ 代表每个 block 中可以开启的最大线程数量；SM_{size} 代表共享内存的大小。

在长度为 L 的 s _ cust 上分配大小为 m 的滑动门，block 内的线程按照线程号分配其计算的数据。在同一时刻，block 内所有线程的存储数据均或落在滑动门范围内，且每个线程会计算独立维度的数据，这样就避免了存储数据时的线程同步。并行计算完 m 个数据之后，滑动门会向前移动，同时线程也会向前启动，计算下一批 m 个数据，直到滑动门回到初始位置为止。滑动门并行计算中心点的算法过程伪代码如下：

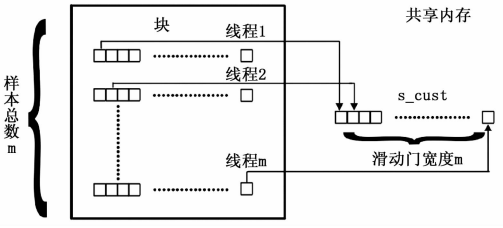

图 4　滑动门并行计算中心点

输入：所有样本的聚类结果；总数为 n 的样本集 D，其中样本为 L 维向量；簇数目 k。

输出：所有簇的中心点集合 U

for 每个簇 do

1）根据（4）选取合适的 m 值。

2）for j = n_i ,do：

（1）j = j/m

（2）for i=0 to L

for 每个线程 paraplle do：

　s=blockIdx. x × block_Size + (theadIdx. x +i)%L;

s_cust[i] += data[s];

end loop;

（3）s=blockIdx. x × block_Size + i;

　data[blockIdx. x × L + i]=s_cust[i];

　end loop;

3）or 每个线程 paralle do：

U[threadIdx. x] = data[0][threadIdx. x] / ;

end loop

根据实验环境的 GPU 配置，由（1）可以计算出最合适的 m 值。通过输入的总数为 n 的样本集，可以计算出程序所需的迭代次数为 $L \times \log_m n$。其中，步骤（2）的迭代次数是 L，m 个维的值将会通过滑动门的方式被并行计算。同一个块中所有

（下转第 279 页）