· 先进计算与数据处理 ·　　文章编号: 1000-3428(2018)06-0001-07　　文献标志码: A　　中图分类号: TP391

一种基于 GPU 的 KNN 动态扩展查询策略

汤　佳,龚奕利,李文海

(武汉大学 计算机学院 软件工程国家重点实验室,武汉 430072)

摘　要: 传统的图形处理器(GPU)执行 PGrid 索引 K 最近邻(KNN)查询方法时存在查询粒度大、冗余计算多、性能不稳定等问题。为此,基于空间 KNN 关系查询,提出一种基于细粒度划分查找范围的 KNN 查询策略。基于欧氏距离的三角不等特性构建 Cell 的动态查询范围扩展,实现查询范围相对于 Cell 各个边界距离的细粒度划分和扩展,分析给定 K 值时对象数量的优化格网尺度。实验结果表明,与传统 KNN 查询方法相比,该查询策略在不同 K 值和格网划分尺度下具有明显的性能优势。

关键词: 图形处理器;计算统一设备架构;测试指标;K 最近邻查询;格网索引

中文引用格式:汤　佳,龚奕利,李文海. 一种基于 GPU 的 KNN 动态扩展查询策略[J]. 计算机工程,2018,44(6):1-7.

英文引用格式:TANG Jia,GONG Yili,LI Wenhai. A KNN dynamic extended query strategy based on GPU[J]. Computer Engineering,2018,44(6):1-7.

A KNN Dynamic Extended Query Strategy Based on GPU

TANG Jia,GONG Yili,LI Wenhai

(State Key Laboratory of Software Engineering,Computer School,Wuhan University,Wuhan 430072,China)

【**Abstract**】Traditional Graphics Processing Unit (GPU) executing PGrid index K-Nearest Neighbor (KNN) query method has problems such as large query granularity, redundant calculation, and unstable performance. Therefore, facing spatial KNN relation queries, a KNN query strategy based on fine-grained partition search is proposed. Based on the Euclidean distance trigonometric inequality feature, the dynamic query range expansion based on Cell is constructed to realize the fine-grained partition and expansion of the query range with respect to each boundary distance of the Cell, and the optimum grid scales of object numbers for a given K value scale are theoretically analyzesd. Experimental results show that compared with the traditional KNN query method, the query strategy has obvious performance advantages under different K values and grid division scales.

【**Key words**】Graphics Processing Unit (GPU); Compute Unified Device Architecture (CUDA); test indicators; K-Nearest Neighbor (KNN) query; grid index

DOI:10. 19678/j. issn. 1000-3428. 0047069

0　概述

随数据挖掘和深度学习的日益广泛应用,基于相似度的样本排序和邻域查询需求与日俱增,亟需高效手段快速处理大数据的相对距离查询。其中,典型的应用场景为基于位置的服务(Location-Based Service, LBS)中 K 最近邻(k-Nearest Neighbor, KNN)查询位置查询。随着对象数量的快速增长,图形处理器(Graphics Processing Unit, GPU)下 KNN 位置查询的时间复杂度受制于有限的 CPU 计算能力和高昂的更新同步负载,因此,GPU 日益受到重视。

研究基于 PGrid 在 GPU 上探讨 KNN 的查询优化方法问题,PGrid[1] 是一种主存索引结构,它基于空间分区的思想,可支持对象快速查询,且易于发挥现代处理器的卓越性能,被广泛应用于 LBS 应用中。PGrid 中基于对象位置的常见查询有两种:范围查询和 KNN 查询。范围查询即给出一个查找范围然后返回范围内的全部对象;KNN 查询查找并返回指定对象周围最近的 k 个对象,由于实际应用中范围阈值所代表的意义常难以确定,因此 KNN 查询较之范围查询有着更实用的价值。

GPU 因强大的计算能力和随着计算统一设备架构(Compute Unified Device Architecture, CUDA)平台发展而带来的易用性,被广泛用于解决复杂的计算问题。在内存局部性较好的上下文中,GPU

作者简介:汤　佳(1991—),男,硕士,主研方向为并行计算;龚奕利、李文海,副教授。

收稿日期:2017-05-04　　修回日期:2017-06-21　　**E-mail**:lwh@whu.edu.cn

较 CPU 有显著的优势[2]。然而,传统的 GPU 执行 PGrid 索引 KNN 查询的方法存在查询粒度大、冗余计算多、性能不稳定的缺点,也即在每次完成一轮查询且未完全返回 K 个对象时,下一轮查询直接扩充一圈 Cell,当 K 值较大时,传统 KNN 可以快速扩充到与 K 值相匹配的查询范围,但当 K 值较小时,余下最近的对象仅存在所扩充的极少数 Cell 内,冗余计算量巨大,故在查询 K 值较小或格网划分尺度较小时特别值得深入研究。如何在保证 GPU 显存访问局部性的前提下,对 KNN 这类相对距离查询提供高效的剪枝优化手段意义重大。

本文提出一种细粒度划分查找范围的 KNN 查询策略 Fg-KNN(Fine-grained KNN),基于欧氏距离的三角不等特性实现 Cell 的动态查询范围扩展,实现查询范围相对于 Cell 各个边界距离的细粒度划分和扩展。在此基础上,对给定 K 值和对象数量的优化格网尺度进行理论分析,研究基于相对距离剪枝的动态执行策略适用于其他具有三角不等特性的公制距离 KNN 查询。

1 相关工作

1.1 GPU 计算资源构成

用于通用计算的 GPU 是 NVIDIA 公司开发的一种高性能计算部件,经过几代的架构发展从而提供越来越强大的计算能力和并行度。典型的 Kepler 架构 GK110 有 15 个流处理器(Stream Multiprocessor, SMX),每个流处理器含 192 个 CUDA 核心(CUDA core),总计 2 880 个 CUDA 核;显存容量 12 GB,带宽达 288 Gb/s,每个流处理器配备 48 KB 的共享内存(Shared Memory)供线程组使用,每个 CUDA 核心拥有 255 个寄存器文件(Register File)供显存私有访问。该架构不仅在硬件上配备强大的计算能力和带宽速度,而且引入如动态并行化和 Hyper-Q 等编程特性,极大促进了 GPU 运行并行程序的速度和优化效率。

1.2 空间 KNN 查询

基于空间分区的内存索引结构 PGrid 由文献[1]提出,这种格网结构易于发挥现代处理器的并行能力。并且对更新操作和查询操作保持一份索引,能够为查询请求返回最新的对象信息。它将更新操作视为不可分割的整体,而非删除操作和插入操作的结合,可以防止查询时发生对象消失。

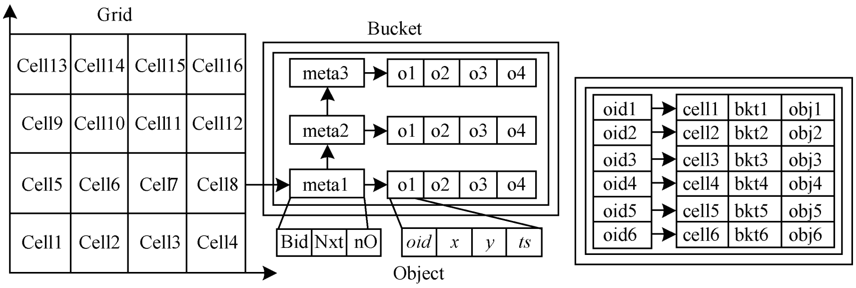
KNN 查询是 PGrid 索引中一种重要的查询方式。随着互联网数据的不断增长,基于对象时间和空间地理位置的 KNN 查询[3]得到发展,大量关于 KNN 查询性能分析[4]、性能优化[5-6]以及异常检测[7]相关的研究开始涌现。文献[8]提出了传统的基于格网索引的 KNN 查询算法,此算法部分分析查询对象在 Cell 内的位置,并采用一次一圈的粗粒度扩充查询范围的思路,存在冗余计算多、性能不稳定的缺点。基于此,本文提出基于查询范围扩充,根据对象与 Cell 4 条边界距离进行扩充粒度更细的 Fg-KNN 查询方法,从而减少冗余计算。

除了格网索引,文献[9]提出基于四叉树和 GPU 的 KNN 查询算法,该算法执行过程需要不断遍历整个四叉树,其性能受对象个数影响严重;而基于格网索引的 KNN 查询依靠两级索引结构免去对象的遍历。当 K 为 128、对象个数为 50 万时,基于四叉树的 KNN 查询算法用时 42 s 完成全部对象的查找。在本文基于 PGrid 索引的 fg-KNN 方法中,对象信息的获取无须遍历整个索引结构,算法性能受对象个数影响较小,从而在千万级数量的对象数时仍然保持高性能。

计算型 GPU 拥有强大的性能和编程便利性,越来越多地用于处理复杂计算问题,如图算法[2,10]、数据库更新查询[11]、解密算法[12-13]、遥感数据分析[14]、矩阵计算[15]等。

2 基于 PGrid 的 KNN 查询

PGrid 是一种能够广泛应用于基于位置的服务中的内存索引,图 1 描述了 PGrid 的两级索引结构。图 1(a)所示为存储对象地理位置信息的 PGrid 一级索引,使用一个格网并横竖划分为若干个 Cell,每个 Cell 内的所有对象信息存放在该 Cell 的 Bucket 列表中,该 Cell 含有一个指针(头指针)指向该 Bucket 列表的第一个 Bucket(头 Bucket);Bucket 之间通过指针相连,每个 Bucket 含有一个元节点(meta)指向固定数量的对象数组,meta 含有 3 个元素:Bid,Nxt,nO。因为异构 PGrid 索引中 Bucket 提前分配,每个 Bucket 都有一个全局唯一的 Bucket 号 Bid;Nxt 指向当前 Bucket 的下一个 Bucket;对象信息使用四元组(oid,x,y,ts)描述:oid 为对象号,x 和 y 表示对象的横、纵坐标,ts 表示时间戳。图 1(b)为 PGrid 二级索引,通过一个哈希表存储对象在一级索引中的位置,哈希表的键为 oid 指明具体对象,值为一个三元组,其中,Cell 表示该对象处于格网的第几个 Cell,bkt 表示该对象处于该 Cell 的第几个 Bucket 中,obj 表示该对象处于该 Bucket 的对象数组的第几个位置,如此可通过二级索引找到对象在一级索引中位置。

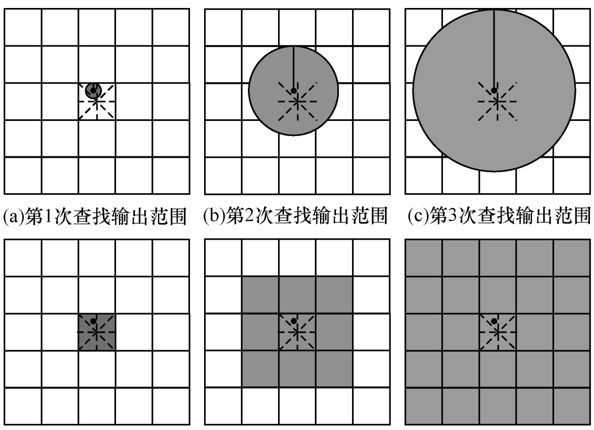


(a)一级索引　　(b)二级索引

图 1　PGrid 索引结构

PGrid 索引中的 KNN 查询请求可以用二元组(qid,oid)进行描述,其中,qid 表示查询请求编号,oid 表示查询的对象,查询结果需要返回该对象地理位置附近最近的 K 个对象。暴力搜索是最直接、同时也最低效的查找算法,其性能受对象个数影响,在对象数较多时效率低下。

文献[6]采用的传统 KNN 查询采用部分分析对象位置并逐层扩展查询范围,最终找到距查询对象最近的 K 个对象的方法。传统 KNN 查询执行过程如图 2 所示。首先计算查询对象与所处 Cell 4 条边的最短距离(设为 r);然后进行第 1 次查找,第 1 次查找的输出范围为以该对象为圆心、以 r 为半径的圆,查找范围为该 Cell;若第 1 次查找不成功则进行第 2 次查找,第 2 次查找的输出范围为以该对象为圆心、以 $r+1$ 为半径的圆,查找范围向外扩充一圈为 9 个 Cell;若第 2 次查找不成功则进行第 3 次查找,第 3 次查找的输出范围为以该对象为圆心、以 $r+2$ 为半径的圆,查找范围继续向外扩充一圈为 25 个Cell;如此循环,第 N 次查找的输出范围为以该对象为圆心、以 $r+N-1$ 为半径的圆,查找范围经扩充为$(2\times N-1)^2$ 个 Cell,直到寻找到最近的 K 个对象结束查询。



(a)第1次查找输出范围　(b)第2次查找输出范围　(c)第3次查找输出范围

(d)第1次查找范围　(e)第2次查找范围　(f)第3次查找范围

图 2　基于格网索引的传统 KNN 查询过程

传统 KNN 查询依据查询对象距所处 Cell 的最短边界距离逐层向外圈 Cell 扩展查询范围,而没有综合考虑对象距所处 Cell 的 4 条边界距离进行更细粒度的查询范围扩充,从而造成冗余计算过多,影响 GPU 执行效率。

3　相对距离自适应扩展

本文提出一种基于相对位置的动态范围扩展策略,基于最大最小距离进行细粒度查询范围扩充,提高 KNN 查询效率。

3.1　边界相对距离

基于对象距所处 Cell 4 条边的边界距离将所属 Cell 的范围划分为 8 个部分,如图 3 所示。查询对象必然属于该 Cell 的 8 个部分空间的其中一部分;设对象距所处 Cell 的 X 轴距离为 a,距 Y 轴距离为 b,则当 $a<0.5$、$b<0.5$ 且 $a<b$ 时对象位于第一部分,当 $a<0.5$、$b<0.5$ 且 $a>b$ 时对象位于第二部分,依次类推。基于对象所处的 Cell 空间位置,可进一步将该 Cell 与其周围的 Cell 进行查找过程的排序。

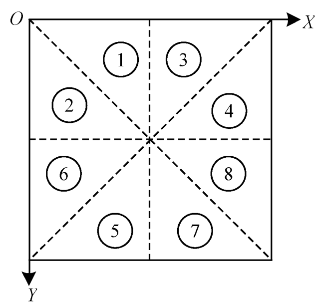
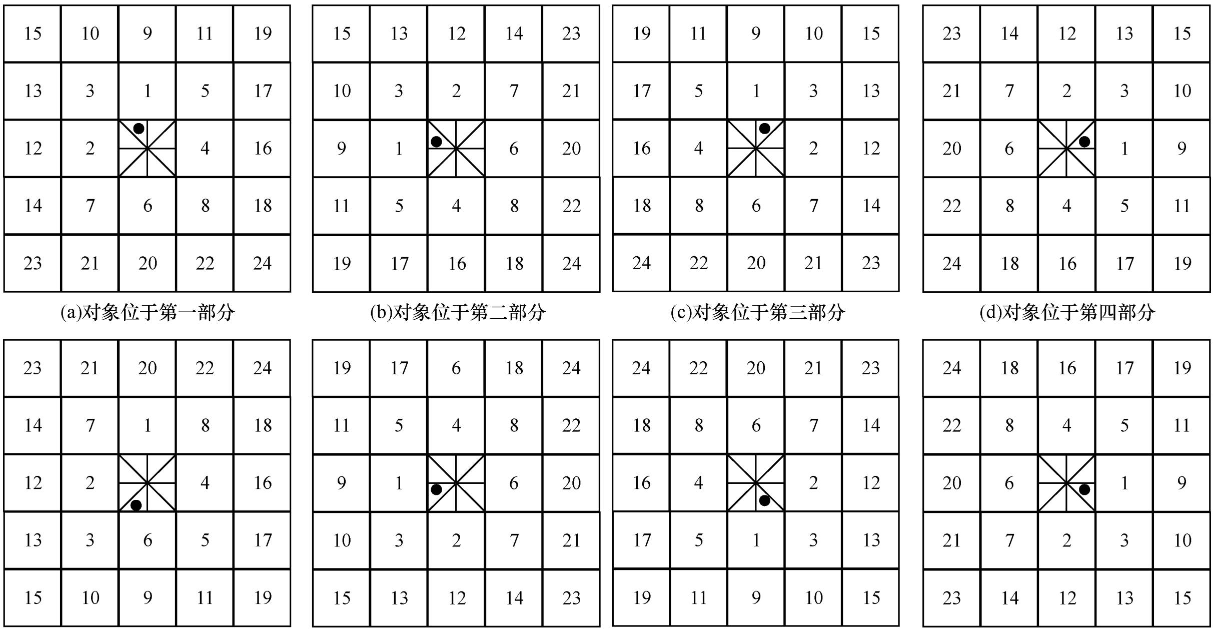


图 3　Cell 空间划分示意图

如图 4 所示为对象处于 Cell 不同部分时,该对象所处 Cell 周围的 5 个 ×5 个 Cell 的 KNN 查询次序。距离对象较近的 Cell 被优先查找,距离对象较远的对象被推后查找。



(a)对象位于第一部分 (b)对象位于第二部分 (c)对象位于第三部分 (d)对象位于第四部分

(e)对象位于第五部分 (f)对象位于第六部分 (g)对象位于第七部分 (h)对象位于第八部分

图 4 周围 Cell 的查找次序

3.2 动态扩展策略

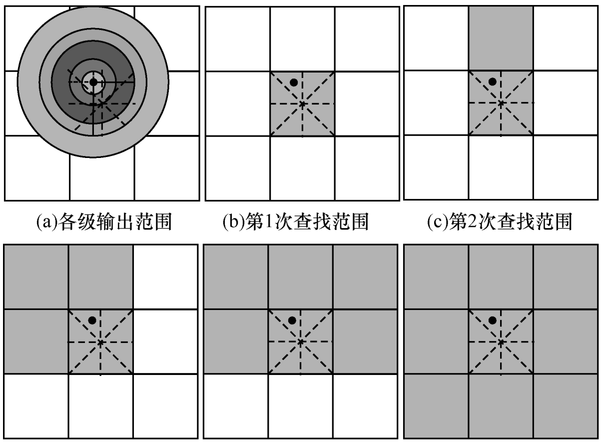
本节基于查询范围动态扩展进行 KNN 查询策略优化。优化核心流程可描述如下：

1）迭代执行查询直至 K 得到满足；

2）每一次查找产生查找范围和输出范围，并且输出查找范围内属于输出范围的对象，然后判断是否满足 K 个对象；

3）依据当前有效查询结果数量，决定是否需要进行下一次查找，若是则转第 1）步。

图 5 给出一个动态扩展的示例，与传统 KNN 查询不同，图 5 采用了基于 Cell 相对距离的查找范围扩展。其中多级输出范围分别是以查询对象为圆心、以该对象到 Cell 边界的距离为半径的同心圆，图 5根据对象位置决定所属 Cell 及周围的 8 个 Cell 的查找次序，该 9 个 Cell 和图 4 中的四级输出范围具有固定的对应关系。


(a)各级输出范围 (b)第1次查找范围 (c)第2次查找范围

(d)第3次查找范围 (e)第4次查找范围 (f)第5次查找范围

图 5 KNN 多级查询的输出范围和查找范围

第 1 次输出范围对应的查找范围为图 4 中的 1 号Cell；图 5 中第 2 次输出范围对应的查找范围为图 4 中的 1 号、2 号 Cell；图 5 中第 3 次输出范围对应的查找范围为图 4 中的 1 号 ~4 号 Cell；图 5 中第 4 次输出范围对应的查找范围为图 4 中的 1 号 ~ 6 号Cell。在第 1 次查找时，输出范围是以对象为圆心、以对象距边界最短的边缘距离为半径的圆（图 5(a) 中 1 号圆），查找范围为当前 Cell，查询会遍历当前 Cell 点全部对象并计算其与查询对象的距离，然后判断该对象是否属于输出范围，若属于则写入到此查询的查询结果。

第 1 次查找未完成查询时进行第 2 次查找，此时输出范围是以对象为圆心、以对象距边界第二短的边缘距离为半径的圆（图 5(a) 中 2 号圆），查询遍历 1 号、2 号 Cell 的全部对象并比较该对象与查询对象的距离是否小于第二短边缘距离，并且大于最短边缘距离（距离小于等于最短边缘距离的对象已在第 1 次查找中返回），如成立则写入查询结果。

第 2 次查找未完成时进行第 3 次查找，此时输出范围是以对象为圆心、以对象距边界第三短的边缘距离为半径的圆（图 5(a) 中 3 号圆），查询遍历1 号 ~4 号 Cell 的全部对象并比较该对象与查询对象的距离是否小于第三短边缘距离，并且大于第二短边缘距离（距离小于等于第二短边缘距离的对象已在第 2 次查找中返回），若成立则写入查询结果。

第 3 次查找未完成时进行第 4 次查找，此时此时输出范围是以对象为圆心、以对象距边界最长的边缘距离为半径的圆（图 5(a) 中 4 号圆），查询遍历 1 号 ~6 号 Cell 的全部对象并比较该对象与查询对

象的距离是否小于最长边缘距离,并且大于第三短边缘距离(距离小于等于第三短边缘距离的对象已在第 2 次查找中返回),若成立则写入查询结果。

按此步骤细粒度扩充查找范围,直到完成查询。

算法　Fg-KNN 查询

输入　查询请求 Query;格网一级索引 Grid;格网二级索引 SecIndex

输出　距查询对象最近的 K 个对象

```
1. oid ←Query. oid;  //查询对象对象号
2. sie ← SecIndex → indexes[ oid ];  //查询对象的//二级索引
3. object ←Grid. GetObject( sie);  //查询对象
4. cell ← Grid → cells[ sie . idx _ cell ];  //查询对象//所属 Cell
5. 计算 object 距 cell 四条边界的距离 d1,d2,d3,d4(其中 d1≤d2≤d3≤d4);
6. while 查询未找到 K 个最近对象 do
7.      r ←d1;
8.      执行以对象为圆心、以 r 为半径的范围查找;
9.      if 已找到 K 个最近对象 then
10.         break;
11.     r ←d2;
12.     执行以对象为圆心、以 r 为半径的范围查找;
13.     if 已找到 K 个最近对象 then
14.         break;
15.     r ←d3;
16.     执行以对象为圆心、以 r 为半径的范围查找;
17.     if 已找到 K 个最近对象 then
18.         break;
19.     r ←d4;
20.     执行以对象为圆心、以 r 为半径的范围查找;
21.     d1 ←d1 +1;d2 ←d2 +1;d3 ←d3 +1;d4 ←d4 +1;
22. end
```

依据 KNN 查询的策略和执行过程,上述算法描述了 Fg-KNN 查询算法的实现,根据输入参数格网索引 Grid 和 SecIndex,计算出距查询请求 Query 所查对象最近的 K 个对象并输出。其中,第 1 行、第 2 行得到查询对象的对象号和二级索引,第 3 行、第 4 行根据对象号和格网索引得到该查询对象和所处 Cell,第 5 行计算查询对象距所属 Cell 的边界距离并排序,第 6 行 ~ 第 22 行为查询的执行过程,在以查询对象为圆心,其半径基于 4 条边界距离递增的圆内进行最近邻对象的查找。

无论是传统的 KNN 查询还是上述算法描述的 Fg-KNN 查询,在查询范围从小到大扩充过程中,范围内 Cell 所包含对象个数越来越多,当对象个数达到(大于或等于)k 个时,查询成功,因此,算法时间复杂度为 $O(K)$。

3.3　Grid 优化分析

本节针对给定 k 值,从理论上分析最佳的 Fg-KNN算法格网划分尺度,并通过实验结果进行验证。

Fg-KNN 查询算法的执行时间受 2 个因素影响:查找次数和每次查找用时。若想获得最短执行时间,则需要最大程度地降低查找次数和每次查找用时。首先将查找次数降至最低,即查找次数为 1,然后分析 1 次查找结束查询的最少用时。

分析对象在所处 Cell 内的平均第一次输出范围大小。输出范围为一个圆,设对象所处 Cell 边长为 L,Cell 内对象个数为 m,该圆的半径取值为[0,$L/2$],则平均半径为:

$$\int_0^{\frac{L}{2}} x\mathrm{d}x = \frac{L}{4}$$

如果查询过程一次完成,则可在半径为 $L/4$ 的输出范围内找到 K 个对象并返回,有:

$$\frac{K}{m} = \frac{\pi \times \left(\frac{L}{4}\right)^2}{L^2}$$

即:

$$\frac{K}{m} = \frac{\pi}{16}$$

然而一次完成查询并不是最优的,可以选择将格网划分尺度进一步细分从而 2 次完成查询。假定在原格网划分尺度下一次查询用时为 T_1,则将格网进一步细分,2 次查找完成查询的用时为 T_2,其中,第 1 次查找 1 个 Cell,第 2 次查找 2 个 Cell,每个 Cell 大小为细分前的 1/4,有:

$$T_2 = 1 \times \frac{T_1}{4} + 2 \times \frac{T_1}{4} = \frac{3}{4}T_1$$

因此,后者所用时间更少。

如给定 7 个 ~10 个查询对象,当 $K = 100$ 时,有:

$$m = \frac{16 \times K}{\pi} = \frac{16 \times 100}{3.14} \approx 510$$

在格网以 128 ×128 划分时每个 Cell 所含对象数与此接近,故 256 ×256 是最优的格网划分尺度。

4　实验结果与分析

为说明本文所提查询策略的有效性,分别在 PGrid 查询的不同对象分布方式、不同的 K 取值时进行实验。实验中 PGrid 框架包含具有不同分布的 1 000 万个对象,这 1 000 万个对象的位置信息是通过均匀分布或高斯分布生成函数随机生成的;格网以32 ×32 ~ 1 024 × 1 024 分割为 Cell。实验执行 100 万 KNN查询请求,按分段处理每段 4 000 个查询运行25 次循环的时间,单位为 s。

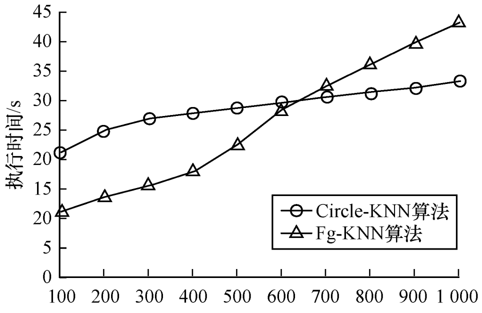
实验所使用的服务器主机端配置为双四核 Intel Xeon E5-2650 CPU,主频为 2.4 GHz,内存 16 GB;设备端 GPU 型号为 NVIDIA Tesla K40c,其内置一颗 Kepler GK110 芯片,显存 12 GB,支持 PCI-E3.0,显存带宽约 288 Gb/s。KNN 查询程序由 CUDA-8.0 编译并运行在 CentOS-7.2(内核 linux-3.10.0)环境中。

传统 KNN 查询逐层扩充查找范围的思路,实验部分称之为 Circle-KNN;本文提出的称之为 Fg-KNN(Fine grained KNN)。

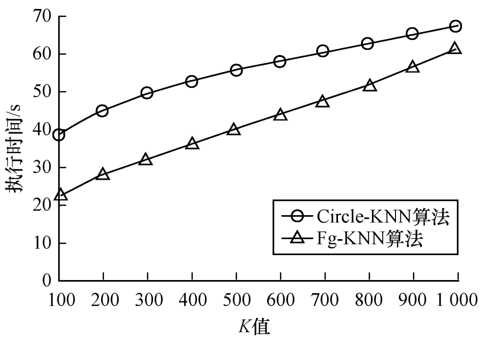
实验过程分 3 个部分进行,分别在不同 K 值和不同格网尺度下比较 Circle-KNN 和 Fg-KNN 的性能,以及给出针对给定 K 值来选择最佳格网划分的适应性分析。

4.1　K 值对 2 种策略的影响

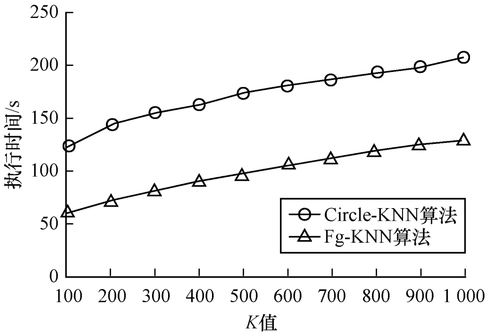
本文实验 PGrid 的格网尺度为 128 × 128,测试不同 K 值大小(从 100 到 1 000)对 Circle-KNN 和 Fg-KNN 执行效率的影响。采用均匀分布、高斯分布 10 核、高斯分布 100 核的数据分布类别进行实验,结果如图 6 所示。



(a)均匀分布



(b)高斯分布10核



(c)高斯分布100核

图 6　不同 K 值时的查询时间

当增大 K 值时,查询需要返回的对象数增多,查询过程更迟结束,故 Circle-KNN 查询和 Fg-KNN 查询性能皆下降。

从图 6 结果可知,除图 6(a)的对象均匀分布且 K 值较大(大于 700)时,Circle-KNN 性能优于 Fg-KNN,这是因为当对象均匀分布时,每个 Cell 内所含有的对象数几乎相同,且当 K 值较大时,KNN 查询需要遍历足够多的 Cell,才能够返回最近邻的 K 个对象,因此,细粒度查询范围扩充的 Fg-KNN 需要进行比 Circle-KNN 更多轮数的查询,故所用时间更长。在其他所有对象分布模式或 K 值大小下,Fg-KNN 查询相交 Circle-KNN 查询皆取得一定性能优势,最好情况下(对象高斯分布 100 核且 K 值为 100)前者是后者性能的 2.02 倍。

128 × 128 的格网划分属于中等尺度的格网划分,此时每个 Cell 内含有的对象数约为 600。在完成查询所需的查找此数较少(需要 K 值相对较小)时,Fg-KNN 所需遍历的 Cell 个数明显较少,因此效率更高。在 K 值较大时,Circle-KNN 能够在较少的查找次数下完成查询,而 Fg-KNN 需要首先进行多次不完整的查找,故性能稍差。

4.2　格网尺度

本文实验固定对象分布方式为均匀分布,测试不同格网划分尺度(从 32 × 32 到 1 024 × 1 024)对 Circle-KNN 和 Fg-KNN 查询执行效率的影响。在 K 值分别取 50、100、150 时进行实验。

固定个数的对象分布于格网内,当格网以较小尺度划分时,每个 Cell 内的平均对象个数较多;当格网以较大尺度划分时,每个 Cell 内的平均对象个数较少。

如果 K 值较小,使得无论是在 Circle-KNN 算法中还是在 Fg-KNN 算法中,查询都能够在第 1 次查找的单个 Cell 内找到距离查询对象最近的 K 个对象。1 000 个万对象分布在格网内,当格网以 32 × 32 划分时,平均每个 Cell 约含 10 000 个对象,此时 Circle-KNN 查询和 Fg-KNN 查询必定能够在第 1 次查找的单个 Cell 内找到距离查询对象最近的 K 个对象,适当增加格网尺度可减少该 Cell 内的对象数以及无用的比较次数:当尺度从 32 × 32 增大到 256 × 256 时,2 种算法性能都得到提升。

而图 7(a)和图 7(b)、图 7(c)不同的是,当格网划分尺度从 256 × 256 增大到 512 × 512 时,图 7(a)中 Fg-KNN 的性能继续提升,图 7(b)、图 7(c)的性能稍微下降。当格网以 256 × 256 划分时,平均每个 Cell 约含 150 个对象,当格网以 512 × 512 划分时,平均每个 Cell 约含 40 个对象,因此 $K = 50$ 且格网划分尺度从 256 × 256 增大到 512 × 512 时,Fg-KNN 虽不能一次查找完成查询,但它可 2 次查找完成查询,且格网尺度增大,每个 Cell 所含对象数减少,没此查找所用时间减少,故性能可进一步提升;$K = 100$ 或 $K = 150$ 结果类似。

相比 Fg-KNN 查询,Circle-KNN 查询是一种粗粒度扩充查询范围的方式,因此,在 K 值较小或格网划分尺度较小时,其性能较差。

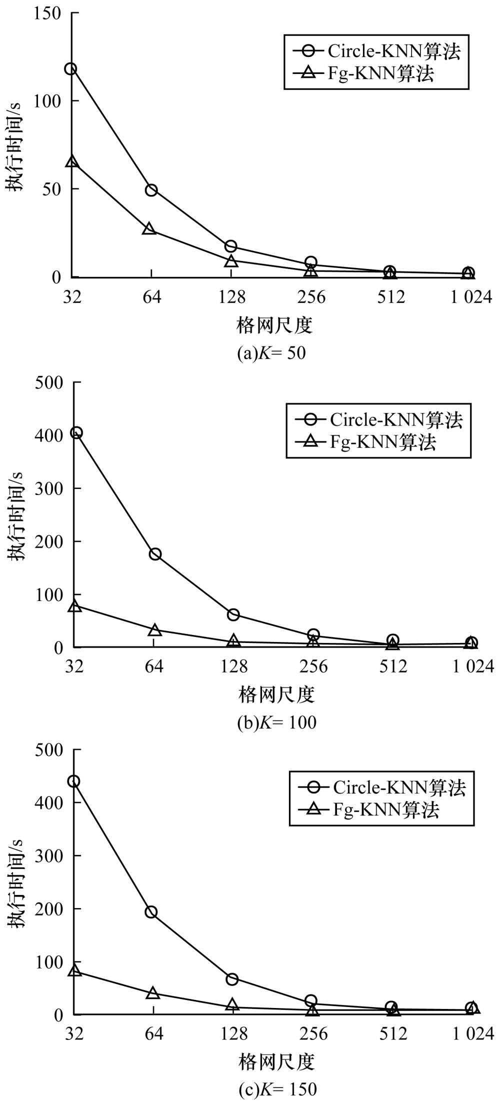


图 7　不同格网划分尺度下的执行时间

4.3　适应性验证

表 1 为当 $K=100$ 时不同格网划分尺度下 Fg-KNN 查询的执行结果,发现执行时间存在鞍点,且鞍点 256 × 256 是最佳格网划分,这与在 3.3 节中的分析一致。

表 1　不同格网尺度时 Fg-KNN 查询时间

格网尺度	Fg-KNN/s
32	76.50
64	33.30
128	11.70
256	5.04
512	5.19
1 024	5.30

5　结束语

本文提出一种细粒度划分查找范围的 KNN 查询策略。该查询策略依据对象在所处 Cell 内的位置采用更加细粒度的查询范围扩充,使得 KNN 查询在查找成功时尽早返回,最大程度地减少冗余计算和对比,从而节省执行时间,并从理论上分析给定 K 值时最优的格网划分尺度。实验结果表明,该查询策略在不同 K 值和格网划分尺度下较传统 KNN 方法具有明显的性能优势。

参考文献

[1] SIDLAUSKAS D,SALTENIS S,JENSEN C S. Parallel main-memory indexing for moving-object query and update workload[C]//Proceedings of SIGMOD'12. Scottsdale,USA:IEEE Press,2012:37-48.

[2] BUSATO F, BOMBIERI N. BFS-4K: an efficient implementation of BFS for kepler GPU architectures[J]. IEEE Transactions on Parallel & Distributed Systems, 2015,26(7):1826-1838.

[3] 李　晨,申德荣,朱命冬,等. 一种对时空信息的 KNN 查询处理方法[J]. 软件学报,2016,27(9):2278-2289.

[4] 刘　灿,张德贤. KNN 查询处理算法性能研究[J]. 苏州科技学院学报(自然科学版),2006,23(3):73-77.

[5] 梁俊杰,李凤华,刘琼妮,等. MapReduce 框架下的优化高维索引与 KNN 查询[J]. 电子学报,2016,44(8):1873-1880.

[6] 蔚赵春. 无线传感器网络中自适应数据存储与 KNN 查询处理研究[D]. 上海:复旦大学,2008.

[7] 卢　鋆,吴忠望,王　宇,等. 基于 kNN 算法的异常行为检测方法研究[J]. 计算机工程,2007,33(7):133-134.

[8] WARD P G, HE Z, ZHANG R, et al. Real-time continuous intersection joins over large sets of moving objects using graphic processing units[J]. The VLDB Journal,2014,23(6):965-985.

[9] LETTICH F,ORLANDO S,SILVESTRI C. Processing streams of spatial k-NN queries and position updates on manycore GPUs[C]//Proceedings of SIGSPATIAL International Conference on Advances in Geographic Information Systems. Washington D. C., USA: IEEE Press,2015:26.

[10] 郭绍忠,王　伟,周　刚,等. 基于 GPU 的单源最短路径算法设计与实现[J]. 计算机工程,2012,38(2):42-44.

[11] 韦春丹,龚奕利,李文海. 一种基于 GPU 的移动对象并行处理框架[J]. 计算机应用与软件,2016,33(10):223-229.

[12] 陈　鹏,曹剑炜,陈庆奎. 基于 GPU 的 H. 264 并行解码算法[J]. 计算机工程,2014,40(1):283-286.

[13] 乐德广,常晋义,刘祥南,等. 基于 GPU 的 MD5 高速解密算法的实现[J]. 计算机工程,2010,36(11):154-155.

[14] 卢　俊,张保明,黄　薇,等. 基于 GPU 的遥感影像数据融合 IHS 变换算法[J]. 计算机工程,2009,35(7):261-263.

[15] 程　豪,张云泉,张先轶,等. CPU-GPU 并行矩阵乘法的实现与性能分析[J]. 计算机工程,2010,36(13):24-26.

编辑　索书志