基于CUDA的并行K-近邻连接算法实现

潘 茜 张育平 陈海燕
(南京航空航天大学计算机科学与技术学院 南京 210016)

摘 要 针对大规模空间数据的K-近邻连接查询问题,设计了一种CUDA编程模型下K-近邻连接算法的并行优化方法。将K-近邻连接算法的并行过程分两个阶段:1)对参与查询的数据集P和Q分别建立R-Tree索引;2)基于R-Tree索引进行KNNJ查询。首先根据结点所在位置划分最小外包框,在CUDA下基于递归网格排序算法创建R-Tree索引。然后在CUDA下基于R-Tree索引进行KNNJ查询,其中涉及并行求距离和并行距离排序两个阶段:求距离阶段利用每一个线程计算任意两点之间的距离,点与点之间距离的求取无依赖并行;排序阶段将快速排序基于CUDA以实现并行化。实验结果表明,随着样本量的不断增大,基于R-Tree索引的并行K-近邻连接算法的优势更加明显,具有高效性和可扩展性。

关键词 CUDA,K-近邻连接,空间查询,并行计算,R-Tree索引
中图法分类号 TP31 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.10.035

Implementation of Parallel K-Nearest Neighbor Join Algorithm Based on CUDA

PAN Qian ZHANG Yu-ping CHEN Hai-yan
(School of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China)

Abstract In order to solve the problem of K-nearest neighbor join query in large scale spatial data, a parallel optimization method of K-nearest neighbor join algorithm based on CUDA programming model was designed. The parallel process of K-nearest neighbor join algorithm is divided into two stages. One is to establish the R-Tree index for the data set Q and P participate in the query, and the other is to carry out the KNNJ query based on R-Tree index. Firstly, MBR is created according to the location of nodes, and the R-Tree index is created based on SRT by CUDA. Then, the KNNJ query is made based on the R-Tree index, including parallel computing and parallel sorting. The distance between two points can be calculated by each thread on the parallel, and quicksort is executed in parallel on the CUDA. Experimental results show that with the increase of sample size, the advantages of parallel K-nearest neighbor algorithm are more obvious, which has high efficiency and scalability.

Keywords CUDA, K-neighbor join, Spatial query, Parallel computing, R-Tree index

1 引言

K-近邻连接(K-Nearest-Neighbor Join,KNNJ)算法作为空间数据查询中最常用的算法之一,已经在KNN分类、K均值聚类、评估计算等方面得到了广泛应用[1],主要用于求取空间关系间的最近邻对象集。但其计算过程相对耗时,随着空间数据量的增大,算法的执行效率明显下降。针对KNNJ查询问题,已经提出了很多方法[2-4],但这些方法在大规模数据的情况下难以扩展。为了使KNNJ算法突破单机单线程的限制,使其在大规模空间数据查询的情况下仍保持一定的高效性和可扩展性,人们越来越多地关注KNNJ算法。KNNJ算法本身具有很强的可并行性,因此采用并行计算可得到很好的效果。

目前,GPU在并行计算方面已取得了显著进展[5]。与CPU的串行设计模式不同,GPU具有天然的并行特性,能提供强于CPU十倍甚至上百倍的性能,极大地促进了计算机图形学及其他应用领域的快速发展。

然而,GPU依然存在一些不足:只能通过图形API来编程;GPU程序不能写入信息到显卡存储区的任何部分;显卡内存与主机内存之间的数据传输带宽成为处理问题的瓶颈[6]。针对这3个问题,NVIDIA公司于2007年推出了一种通用并行计算架构CUDA(Compute Unified Device Architecture)。

CUDA将GPU视为一个数据并行的计算设备,并且无需把这些计算映射到图形API。CUDA以C语言为基础,并对其进行扩展[7]。CUDA已被应用于计算生物学、密码学、计算金融、地理信息系统等众多领域。但另一方面,国内对CUDA的研究起步较晚,大多集中于图像处理方面。本文将CUDA用于KNNJ算法的并行查询。查询过程主要分为两个阶段:1)基于递归网格排序算法(Sort-tile-recursive STR)并行构建R-Tree索引;2)基于R-Tree索引进行并行KNNJ查询,并行查询阶段涉及并行求距离和并行距离排序两部分,

到稿日期:2015-07-07 返修日期:2015-12-13 本文受国家重点基础研究发展计划(973计划)(2014CB744900),南京航空航天大学研究生创新基地开放基金(KFJJ201460)资助。

潘 茜(1990—),女,硕士生,主要研究方向为软件工程,E-mail:420617478@qq.com;张育平(1963—),男,副教授,主要研究方向为软件工程;陈海燕(1978—),女,博士,讲师,主要研究方向为数据挖掘。

第一部分利用每个线程计算一组距离，并行无依赖，第二部分将快速排序法并行化，使得整个查询过程得以并行快速实现。

2 背景知识

2.1 查询定义

本文所采用的距离准则是欧氏距离，表示为 $d(p,q)=(\sum|p_i-q_i|^2)^{\frac{1}{2}}$，对查询的定义如下。

定义 1(KNN 查询) 给定对象 p 和空间对象 $Q=\{q_1,q_2,\cdots,q_N\}$，$KNN(p,Q,K)$ 则为 Q 中距 P 最近的 K 个对象集，即 $KNN(p,Q,k)=\{q_1,q_2,\cdots,q_K\}$。其中，$0\leqslant k\leqslant|Q|$ 且返回的距离值按升序排列。

定义 2(KNNJ 查询)[8] 给定空间关系 $P=\{p_1,p_2,\cdots,p_{NP}\}$，$Q=\{q_1,q_2,\cdots,q_{NQ}\}$，$KNNJ(P,Q,K)$ 返回的是 P 中每个对象的 KNN 查询结果，即 $KNNJ(P,Q,K)=\{p,KNN(p,Q,k)|\text{for all } p\leqslant P\}$。

定义 3(最小外包框) 设函数 $R(p,q)$ 表示 d 维空间中的一个最小外包框(Minimum Bounding Rectangle MBR)，其中 $p=\{p_1,p_2,\cdots,p_d\}$ 为 MBR 中的最小点，$q=\{q_1,q_2,\cdots,q_d\}$ 为 MBR 中的最大点，则对于任意的 $1\leqslant i\leqslant d$，都有 $p_i\leqslant q_i$ 成立。p 即为 MBR 的左下角点，q 为 MBR 的右下角点。

2.2 CUDA 编程模型

CUDA 是一种由 NVIDIA 提出的通用并行计算架构，CUDA 的主要设计目的是：1)扩展标准编程语言，例如 C 语言，使并行算法更加简单的执行。通过使用 CUDA，程序员能更多地专注于算法并行的任务，而不是花很多时间在算法的执行上。2)应用程序通过同时使用 CPU 和 GPU 来支持异构计算。程序串行的部分在 CPU 上执行，并行的部分则在 GPU 上实现。CPU 和 GPU 都有各自的内存空间，因此它们可以同时进行，而不需要竞争内存资源[9]。

2.2.1 Kernel

运行在 GPU 上的程序称为内核函数 Kernel，当被调用时，它会被 N 个不同的 CUDA 线程并行地执行 N 次。Kernel 通过_global_函数类型限定符定义，调用格式为：*Kernel*((Dg,Db,Ns,S))(*param list*)。其中 Dg,Db,Ns,S 被称为执行参数，Dg 是 grid 的维度和尺寸，即一个 grid 有多少个 block；Db 是 block 的维度和尺寸；Ns 是最多动态分配的 shared memory 大小；S 即该函数处于哪个流。每一个线程也被赋予了唯一的 threadID。

2.2.2 线程结构

CUDA 将计算任务映射为大量的可并行执行的线程，并由硬件动态调度和执行。

CUDA 使用 dim3 类型，因此线程可以由一维、二维或三维的线程索引来标识，形成一个一维、二维或三维的线程块。线程索引和 threadID 以一种简单的方式彼此关联。对于一个一维索引，它们是一样的；对于一个大小为(D_x,D_y)的二维 block，线程 *threadID* 是($x+yD_x$)；对于一个大小为(D_x,D_y,D_z)的三维 block，线程 threadID 是($x+yD_x+zD_xD_y$)。

3 基于 CUDA 的并行 KNNJ 算法实现

现假设给定空间关系 $P=\{p_1,p_2,\cdots,p_N\}$ 和 $Q=\{q_1,q_2,\cdots,q_M\}$，KNNJ 的执行过程即先计算对象 p_1 和 Q 之间的 KNN 查询，$p_2,\cdots,p_N$ 与 p_1 类似。共执行了 N 次 KNN 查询，我们以单个对象 p_1 的 KNN 查询为例，深入剖析。

单个对象 p_1 的 KNN 查询过程操作比较繁琐，用 *KNN*[*maxNo*]数组存储 Q 中离 p_1 最近的 K 个值，开始为 0，循环调用 *Dis*()函数，每求得一次两点之间的距离，都要与 *KNN*[*maxNo*]数组中的最大值比较，如果小于数组中的最大值，则将其存入最大值所在位置，以此类推，求得 K 个离 p_1 最近的对象。

在 CUDA 编程模型下，对原始 KNNJ 算法实现并行，称为 MPKJ(Massively Parallel KNN Join)。MPKJ 查询主要分为两个阶段：1)基于 SRT 对参与查询的数据集 P 和 Q 分别建立 R-Tree 索引；2)基于 R-Tree 索引进行 KNNJ 查询。

3.1 CUDA 下 R-Tree 索引的快速构建

在空间对象很少发生变化的情况下，对空间数据进行预处理，创建结构优化的 R-Tree 索引，能很大程度上提高存储空间的利用率。为了提高 R-Tree 索引构建的效率和查询性能，文献[10]提出了 SRT 算法。假设在二维空间中，有 N 个 MBR，每个 R-Tree 结点的最大容量为 M，则 STR 的主要思想是：按 MBR 中点的 x 坐标大小排序，将 N 个 MBR 等分成 S 个组，然后在每一个组中，根据 MBR 中点的 y 坐标大小排序，每 M 个组成一个新的 MBR，依次递归，形成 R-Tree 索引。SRT 排序操作过多，面对海量数据构建效率并不理想。因此，本文在 CUDA 下基于 SRT 创建 R-Tree 索引，实现步骤如下：

1)计算 N 个 MBR 的中心点坐标，并将 x 坐标和 y 坐标的值存于 *center_x*[N]和 *center_y*[N]数组中；

2)创建索引数组 $Index[N]=\{0,1,\cdots,N-1\}$，以 x 坐标值的大小为 *key*，以 *Index* 数组的值为 *value*，使用快速排序对 *center_x*[N]进行升序排列；

3)创建数组 *slice*[N]，用于记录每个索引对应的 MBR 所属的切片号，即 $slice[Index[i]]=i/M(i=0,1,\cdots,N-1)$；

4)与步骤 2)类似，重新将 *Index* 数组置为顺序数组，以 y 坐标值的大小为 *key*，以 *Index* 数组的值为 *value*，使用快速排序对 *center_y*[N]进行排序。

5)以 *Index* 为 *map*，*slice* 为 *value*，进行聚集操作，将结果存于数组 *xSorted*[N]中，即 $xSorted[i]=slice[Index[i]](i=0,1,\cdots,N-1)$；

6)以 *xSorted* 为 *key*，*Index* 的值为 *value*，快速排序；

7)依次将每$\lceil\sqrt{N/M}\rceil$个值从 *Index* 数组中取出，将其对应的 MBR 值合并成新一轮的 MBR；

8)将上一步中的 MBR 作为输入，重复以上步骤，直至只剩一个 MBR 作为根结点。

上述算法，将空间数据按空间范围划分，生成了一棵 R-Tree 索引，有利于基于距离准则的 KNNJ 查询。同时，步骤 4)、6)用两次整体排序代替了对每个切片中的 y 值分别进行排序，不仅方便调用快速排序算法，也满足了 GPU 对并发核函数的要求。

3.2 KNNJ 并行求距离

给定空间中任一对象 p，以及 M 个 MBR，即 $M=\{m_1,m_2,\cdots,m_M\}$，计算 p 到索引中某一结点 r 的距离，即求对象 p 到 M 个 MBR 的距离，处理步骤如下：1)调用 API 函数，将数据存入显存；2)每一个线程计算 p 到任意一个 MBR 的距离值，可表示为：$Dis(p,M)=\{Dis(p,m)|\text{for all } m\in M\}$。

其中，求任意两点间欧氏距离的 Kernel 内核函数定义如下：_global_void dis_parallel (int * data_train,int * data_test, int * dis,int pitch){…}。

3.3 KNNJ 的距离并行排序

1962 年 C. A. R. Hoare 首次提出了快速排序法[11]，在过

去的近半个世纪，快速排序法仍是最实用的算法。它的平均情况的时间复杂度为O(NlogN)。快速排序法采用分治的策略，具有很强的可并行性。

本文将枢轴元素定义成：$pivot = Max/(pow((float)2, level) * 2)$，其中 $level$ 为当前的递归深度，则 $pow((float)2, level)$ 为当前深度子列的个数，Max 则为自定义的最大随机数。对于每一个子列 k，其枢轴元素为 $pivot = pivot * (2 * k+1)$。并行排序过程描述如下：

1)调用 *PreSort*()函数对输入数据进行预处理，计算出所有线程的数据开始和结束的位置。函数定义为：_global_ static void PreSort(…)。

根据具体输入数据量的多少分配足够的块 block，每个 block 都有独立的序列，block 内的每个线程 thread 独立地对数据进行遍历，记录下小于或大于枢轴元素的元素个数，并行执行。遍历结束后，每个 thread 中的数据以从前向后的顺序相加，即将本身数值与之前所有数值之和相加。因此，每个线程都记录了所在 block 中，自己前面所有线程遍历的大于或小于枢轴元素的总和，以此类推。再以这样的方式对 block 进行统计，则最后一个 block 就记录了所有块中大于或小于枢轴元素的个数。由此确定了分界，也就是下次排序时两个数组的始末位置。

2)调用 *Replace*()函数，根据 1)中预处理的结果，将线程遍历到的数据放在相应位置。函数定义为：_global_ static void Replace(…)。

两次遍历需要保证每个线程遍历到的数据都是相同的，线程将遍历到的数值按照之前的统计结果，放在相应的区间中。每个线程都为其前面的线程预留了足够空间，且拥有属于自己的区间，这一过程可以并行执行。等到换位结束，原本输入的数组就被分成了两个子数组，且一个数组中的值都小于等于枢轴元素，另一个数组中的值都大于枢轴元素。定义之后，对两个等大的数组进行位置调换。

3)调用 *FinalSort* 函数进行最终排序。该函数定义成：_global_static void FinalSort(…)。

多次调用之后，原数组被划分成了 2^{LEVEL} 个小的数组，且位于 2^{LEVEL} 个块中，即每个块分到一个子数组。此时再使用多线程并行反而会降低执行效率，因此每块都使用简单的选择排序，单线程执行。

3.4 CUDA 下基于 R-Tree 索引的 KNNJ 查询

KNNJ 查询是以 KNN 算法为基础衍生而来，以基于 R-Tree 索引的 KNN 查询为例，其主要思想是：利用 MBR 的最小距离属性，不断地从最近邻结点中查找最近的对象。若给定空间关系 $Q=\{q_1, q_2, \cdots, q_M\}$，以及 Q 的 R-Tree 索引 B，指定查询点 p，则基于 R-Tree 索引的 KNN 查询算法描述如下：

(1)变量 M 用于存储按最小距离排序之后的索引结点和对应的距离值，变量 K 用于存储最近的 k 个对象；

(2)初始化时，M 中包含查询点 p 到各个根结点的最小距离；

(3)若 M 中元素不为空，并且 K 中对象的个数小于 k，则循环执行以下步骤：

1)从 M 中找到距离最小的结点 r；

①若 r 是索引结点，则将 r 的所有子结点放入 M 中，并记录查询点 p 到各子结点的距离值；

②若 r 是叶结点，则将 r 及其距离值存入 K 中；

2)从 M 中删除结点 r。

若参与空间查询的空间关系为 P，Q，其对应的索引为 A，B，则其 $KNNJ(A,B,K)$ 查询即可表述为：对于 A 中的任一结点 r，若 r 是索引结点，则递归执行 $KNNJ(r,B,K)$；若 r 是叶结点，则执行 $KNN(r,B,K)$。

4 实验例证

本文主要采用的实验平台为 Pentium(R) Dual-Core CPU，主频为 2.00GHz，系统内存为 2GB；显卡采用的是 NVIDIA Tesla C2050，显卡内存为 3GB，显卡的核心频率为 1.15GHz，操作系统是 ubuntu12.04，CUDA 为 5.5 版本。

为了验证 MPKJ 算法的可行性，本文从 3 个方面对其进行实验。现有，空间关系 P 的数据规模为 $N * D$，空间关系 Q 的数据规模为 $M * D$，选用的数据通过(rand()%MAX)随机产生。

(1)设定 $K=5$，$D=8$，$MAX=100$，改变 N 和 M 的大小。记录串行 KNNJ 和 MPKJ 各自所用时间，如图 1 所示，MPKJ 所用时间非常少，在图 1 中很难直观看出来，所以将其更为直观地展示于图 2 中。

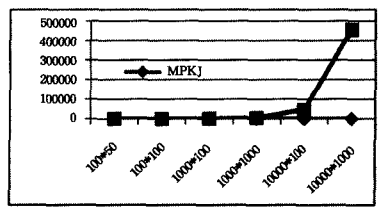

图 1 固定 K、D、MAX，改变 M、N 时 KNNJ 和 MPKJ 所用时间

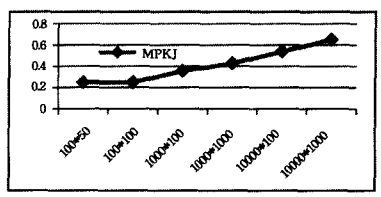

图 2 MPKJ 所用时间

从图 1 中可以清晰地看出，并行后的 KNNJ 算法 MPKJ 随着数据规模的不断增大，计算时间波动非常小，但是串行 KNNJ 增长速度却呈 10 倍趋势上涨，增幅非常大。图 2 中不包含输入显存和输出显存的时间，加上输入输出时间，当 $M=10000$，$N=1000$ 时，MPKJ 所用时间为 1298.25ms，串行 KNNJ 所用时间是其 300 多倍。与其他增速最多只有几十倍的改进算法相比，GPU 并行体现出非常强大的优势。

(2)设定 $M=1000$，$N=1000$，$D=8$，$MAX=100$，改变 K 的值，比较结果如图 3 所示。

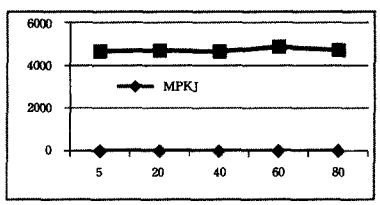

图 3 固定 M、N、D、MAX，改变 K 时 MPKJ 所用时间

从图 3 不难看出，K 值的增大对计算时间并无非常大的影响，这从整个算法层面解释就非常容易了。K 值的大小只与输出到文件中的数据个数有关，而与计算欧氏距离和排序无关，所以其变动与所用时间并无太大关联。

(3)设定 $M=1000$，$N=1000$，$D=8$，$K=5$，改变 MAX 的

(下转第 219 页)

[7] Becker H, Naaman M, Gravano L. Beyond Trending Topics: Real-World Event Identification on Twitter[J]. ICWSM, 2011, 11:438-441

[8] Blei D M, Ng A Y, Jordan M I, et al. Latent Dirichlet allocation [J]. Journal of Machine Learning Research, 2003, 3:993-1022

[9] Blei D M. Introduction to Probabilistic Topic Models[J]. Signal Processing Magazine IEEE, 2011, 27(6):55-65

[10] Ramage D, Dumais S T, Liebling D J. Characterizing Microblogs with Topic Models[J]. ICWSM, 2010, 5(4):130-137

[11] Wang X, McCallum A. Topics over time: a non-Markov continuous-time model of topical trends[C]// Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2006:424-433

[12] Diao Q, Jiang J, Zhu F, et al. Finding bursty topics from microblogs[C]// Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1, Association for Computational Linguistics. 2012:536-544

[13] Heinrich G. Parameter estimation for text analysis[R]. Technical Report, 2004

[14] Xu Ge, Wang Hou-feng. The Development of Topic Models in Natural Language Processing[J]. Chinese Journal of Computer, 2011, 34(8):1423-1436(in Chinese)
徐戈, 王厚峰. 自然语言处理中主题模型的发展[J]. 计算机学报, 2011, 34(8):1423-1436

[15] Shi Jing, Fan Meng, Li Wan-long. Topic Analysis Based on LDA Model[J]. Acta Automatica Sinica, 2009, 35(12):1586-1592(in Chinese)
石晶, 范猛, 李万龙. 基于 LDA 模型的主题分析[J]. 自动化学报, 2009, 35(12):1586-1592

[16] Duan Lian, Guo Wei, Zhu Xin-yan, et al. Constructing Spatio-Temporal Topic Model for Microblog Topic Retrieving[J]. Geomatics and Information Science of Wuhan University, 2014, 39(2):210-213(in Chinese)
段炼, 吴维, 朱欣焰, 等. 基于时空主题模型的微博主题提取[J]. 武汉大学学报(信息科学版), 2014, 39(2):210-213

[17] Chen Wen-tao, Zhang Xiao-ming, Li Zhou-jun. Analysis of Topic Models on Modeling MicroBlog User Interestringness[J]. Computer Science, 2013, 40(4):127-130(in Chinese)
陈文涛, 张小明, 李舟军. 构建微博用户兴趣模型的主题模型的分析[J]. 计算机科学, 2013, 40(4):127-130

(上接第 192 页)

大小，比较结果如图 4 所示。

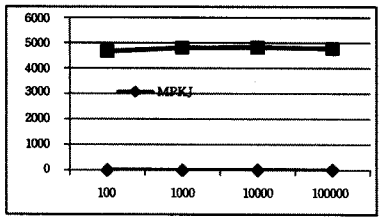

图 4 固定 M、N、D、K，改变 MAX 时 MPKJ 所用时间

从图 4 中可以明显看出，MAX 值的大小与查询时间并无太大关系，MAX 的值只是限定了所产生数据中的最大值，但对于最大值个数等并无限定，所以出现 MAX 增大但计算时间减少的情况，也就不足为奇了。

通过以上实验可以得出结论，随着数据规模的不断增大，串行 KNNJ 所用时间呈十倍以上趋势上升，但 MPKJ 变化缓慢，甚至达到上百倍的加速比，与现有加速比只有几十倍的改进算法相比，优势非常明显，因此 MPKJ 算法具有明显的高效性和可扩展性。

结束语 针对 GPU 的特性，本文设计了一种基于 CUDA 的并行 KNNJ 加速方法 MPKJ，MPKJ 能高效地处理大规模空间数据的查询问题，也可用于其它相似性连接查询问题。实验表明，随着数据规模的不断增大，MPKJ 并行优化的效果也越来越明显，加速比呈成倍增长的趋势。但本文在 CPU 与 GPU 之间的传输问题上并没有过多优化，因此，下一步需要尽可能缩短 CPU 与 GPU 之间的传输时间，进一步优化算法性能，增强 MPKJ 的可扩展性和可适用性。

参 考 文 献

[1] Bohm C, Krebs F. The k-nearest neighbor join: Turbo charging the KDD process[J]. Knowledge Information System, 2004: 6(6):728-749

[2] Xia C Y, Lu H J, Coi B C, et al. Gorder: An efficient method for KDD joins processing[C]// Proc. of the 30th Int'l Conf. on VeryLarge Data Bases. 2004:756-767

[3] Yu C, Cui B, Wang S G, et al. Efficient index-based KNN join processing for high-dimensional data[J]. Information and Software Technology, 2007, 49(4):332-344

[4] Yao B, Li F F, Kumar P. K nearest neighbor queries and KNN-joins in large relational databases (almost) for free[C]// Proc. of the 26th Int'l Conf. on Data Engineering (ICDE). 2010:4-15

[5] Wu En-hua. Technology, current situation and challenge of graphics processor are used for general computing[J]. Journal of Software, 2004, 15(10):1493-1504(in Chinese)
吴恩华. 图形处理器用于通用计算的技术、现状及其挑战[J]. 软件学报, 2004, 15(10):1493-1504

[6] Xu Xue-gui, Zhang Qing. High efficiency parallel remote sensing image processing based on CUDA[J]. Geospatial Information, 2011, 9(6):47-54(in Chinese)
许雪贵, 张清. 基于 CUDA 的高效并行遥感影像处理[J]. 地理空间信息, 2011, 9(6):47-54

[7] Dong Luo, Ge Wan-cheng, Chen Kang-li. Application Research of CUDA parallel computing [J]. Information Technology, 2010, 4(5):11-15(in Chinese)
董荦, 葛万成, 陈康力. CUDA 并行计算的应用研究[J]. 信息技术, 2010, 4(5):11-15

[8] Liu Yi, Jing Ning, Chen Luo, et al. Algorithm for Processing k-Nearest Join Based on R-Tree in MapReduce[J]. Journal of Software, 2013, 24(8):1836-1851(in Chinese)
刘义, 景宁, 陈荦, 等. MapReduce 框架下基于 R-树的 k-近邻连接算法[J]. 软件学报, 2013, 24(8):1836-1851

[9] Sosutha S, Mohana D. Heterogeneous parallel computing using CUDA for chemical process [J]. Procedia Computer Science, 2015, 47(1):237-246

[10] Leutenegger S T, Lopez M A, Edgington J. STR: A Simple and Efficient Algorithm for R-tree Packing [C]// The 13th International Conference on Data Engineering. Birmingham, England, 1997:497-506

[11] Hoare C A R. Quicksort [J]. The Computer J., 1962, 15(1):10-15