DOI:10.13203/j.whugis20130158　　文章编号:1671-8860(2014)09-1068-06

利用 GPU 的 R 树细粒度并行 STR 方法批量构建

邵　华[1]　江　南[1]　胡　斌[1]　吕　恒[1]　朱　进[1]

1　南京师范大学虚拟地理环境教育部重点实验室,江苏 南京,210023

摘　要:大数据时代,需要对海量空间数据更快速地建立高效索引,使用递归排序网格(STR)方法构建的 R 树具有优秀的查询性能,但构建效率不高。本文利用基于计算机图形处理器(GPU)的通用计算具有细粒度可并行性的特点,提出了一种基于 STR 算法的 R 树 GPU 并行构建算法,使用线性数据结构存储 R 树,并且用整体排序代替分段排序,细化算法的并行粒度。实验结果表明,同 CPU 算法相比,本文算法的加速比最高可达 27 倍,并且呈现出随着数据量增大而变大的趋势。本文算法充分利用 GPU 的并行处理能力,高效构建了性能优越的 R 树空间索引。

关键词:R 树;GPU;批量构建;细粒度并行;空间索引

中图法分类号:P208　　**文献标志码**:A

空间索引是对空间数据进行高效管理和分析的基础,针对海量空间数据创建索引结构,兼具了数据密集和计算密集的特点。由于对空间数据进行索引计算存在着数据和计算两方面的天然可并行性,如果使用并行计算技术,将空间索引技术中大量的计算任务分配到多个处理器上,可以获取更多的计算能力,得到更快的处理速度[1]。

R 树索引结构[2]自提出以来,已经成为最流行的多维空间索引结构之一,广泛应用于各种科研及商业空间数据库中,涵盖了地理信息系统(GIS)、超大规模集成电路(VLSI)等领域。许多学者提出了一些衍生变种,比较典型的有 R^+ 树[3]、R* 树[4]、压缩 R 树[5]等。但在这几种 R 树变体的构建中,插入算法代价高,面向大数据量时,创建空间索引的过程效率较低。为此,针对静态空间数据,一些学者提出了 R 树的批量创建技术,在维持或提高查询性能的前提下,一定程度上提高了 R 树的创建效率。

计算机图形处理器(GPU)具有极强的浮点数运算能力,并且计算速度和主存带宽逐年提高。基于 GPU 的通用计算的出现,以及统一计算设备架构(CUDA)和开放运算语言(OpenCL)的提出,大大促进了 GPU 在各个领域的广泛应用[6]。同多核 CPU 或者分布式计算环境相比,GPU 拥有更多的计算核心,从而也要求 GPU 上的算法有更细的并行粒度,这为空间索引技术提高效率带来了新的机遇。文献[7-9]等针对空间查询效率的提高对 GPU 通用计算技术展开了研究,文献[10-11]虽然实现了 GPU 上的 R 树批量构建,但其使用的空间聚类方法简单,建树质量不高,且索引结构创建算法缺少优化。在 R 树的批量创建方法中,递归网格排序算法(sort-tile-recursive, STR)[12]创建的 R 树具有较高的查询效率,并且实现方法相对简单,其算法开销主要集中在排序操作中,而 GPU 的细粒度并行能大大提高对海量数据排序的效率。目前尚未见在 GPU 上使用 STR 方法构建 R 树的相关文献。

1　R 树的批量创建及 STR 方法

在空间对象极少发生变化时,批量创建技术可以通过对数据进行预处理进行全局的聚类分析,建立结构优化的 R 树,降低键树的时间复杂度,并提高存储空间利用率。

文献[5]提出的 Packed R 树和文献[13]提出的 Hilbert Packed R 树分别使用了一维排序以及 Hilbert 分形曲线的思想对空间对象进行聚类,构建的 R 树可以达到接近 100%的空间利用率。

收稿日期:2013-05-22

项目来源:国家科技支撑计划资助项目(2012BAH35B000);国家科技基础条件平台建设项目;江苏高校优势学科建设工程资助项目。

第一作者:邵华,博士生,主要从事高性能空间分析和空间数据挖掘研究。E-mail:shyxiaoxiao@163.com

通讯作者:江南,教授。E-mail:njiang@njnu.edu.cn

为了进一步提高批量构建 R 树的效率及其查询性能,顾及空间对象在二维空间的分布,文献[12]提出了 STR 算法。假设二维空间有 N 个对象的最小外包矩形(MBR),每个 R 树节点的最大容量为 M,STR 方法的基本思想为:按 MBR 中心点的 x 坐标对其排序,并等分成 S 组,在每一组中用 MBR 中心点的 y 坐标进行排序,每 M 个 MBR 形成一个新的节点,并将新节点的 MBR 作为下一次递归的输入,自下而上生成整棵 R 树。

基于 STR 方法创建 R 树的 CPU 算法(C-STR)如下所述。

输入:二维空间对象数据集中的 N 个对象(MBR),每个节点的最大容量 M

输出:包含 N 个空间对象的 R 树索引

1) 将 MBR 按照其中心点的 x 坐标进行排序。

2) 按照上一步的排序结果,将 MBR 在 x 轴方向上分成 $S=\lceil\sqrt{N/M}\rceil$ 个切片,使得每个切片至少包含 $S\cdot M$ 个矩形。

3) 在每一个切片中,对 MBR 用其中心点的 y 坐标进行排序。

4) 在每一个切片中,依次将每 M 个矩形分配为一组,形成一个树节点。

5) 对上一步得到的节点重新计算 MBR,形成上一层的 MBR。

6) 将步骤 5)中的 MBR 输入 1),递归该过程,直至只剩一个 MBR,也就是 R 树的根节点。

STR 方法根据空间对象的空间邻近性,对数据进行全局聚类处理,构建的 R 树具有比较优秀的检索性能。文献[12]指出,对于不同的数据分布类型,不存在一种最优的 R 树,但与 Hilbert R 树相比,STR 方法在空间对象均匀分布或者中度畸变的情况下,能提高 50% 的性能;对于高度畸变的分布,两者性能相当,STR 方法在大多数情况下要优于 Hilbert R 树[14],但是其构建过程中使用多次排序操作,对于海量数据的索引构建效率不高。

在文献[12]之后,也有一些批量创建 R 树的方法被提出,如 TGS 算法[15]、OMT 算法[16]等,但由于没有在创建的开销与查询性能之间取得较好的平衡,并未得到广泛使用。而 STR 方法成为使用最广泛的批量创建 R 树方法之一,应用于一些开源空间数据库中。

2　基于 GPU 的 STR 方法

GPU 计算性能的快速发展及 GPU 编程模式的不断改善,提供了一种提高算法性能的有效途径。本文选择 CUDA 作为本节算法的实现平台,同时该算法也可以较容易地移植至 OpenCL。

2.1　GPU 中 R 树的数据组织

在传统的树结构中,通常使用指针来存储一个节点的子节点,对子节点的访问可能需要两次对主存的访问。而 GPU 中缓存较小,访问主存需要较多的时钟周期,所以在 GPU 中采用指针存储子节点的树结构的访问性能较差。本文采用线性数组的数据结构来表示一棵 R 树,但与文献[11]的方法不同,本文将 MBR 与索引分别存储于两个数组中,由于 STR 方法创建的 R 树是一棵满树(每层最后一个节点可能不满),因此,只要确定了树节点的容量,索引的存储位置隐式地确定该索引在该层中的位置,以及其子节点在下一层的位置。图 1 给出了一棵有 24 个节点,高度为 3 的 R 树在存储器中的表示,索引数组和 MBR 数组分别从 R 树的最底层开始逐层存储,对于每层最后一个可能不满的节点,将其空的子节点位置写为“-1”,冗余只发生在索引数组中,存储 MBR 的数组空间利用率达到了 100%。

考虑建树效率,使用两个线性数组分别独立存储空间对象的 MBR 及其索引,有两个显著的优点:① 通过隐式确定的索引位置,可以直接通过数组下标访问该节点中子节点对应的 MBR,减少内存访问次数;②STR 算法中多次使用了排序算法,是该算法中耗时最多的模块,而将索引和 MBR 独立存储,在对 MBR 排序时,只需对索引数组这样一个整数类型的数组进行对应位置的排序,可以减少内存的重排数据量,提高排序速度。传统的树型数据结构,节点访问效率低且不利于并行构建,采用两个独立线性数组的 R 树数据组织降低了数据结构的空间复杂度,并且通过减少访问节点时的内存地址跳转次数,提高了对节点排序的效率。

2.2　GPU 上的并行 STR 方法

使用 STR 方法构建的 R 树具有优秀的查询性能,但面对大数据量时,使用串行方法构建效率较低,而现有在 GPU 上进行 R 树构建的研究[10-11],为降低构建复杂度,多采用简单的一维排序,构建的 R 树查询性能不高。

GPU 是一种众核处理器,直接将 C-STR 算法的每一个步骤在 GPU 上进行并行化,因为并行的粒度较粗,并不能充分利用硬件资源,需要针对 GPU 的结构特点,研究算法的细粒度并行策略。C-STR算法的主要开销在步骤1)～3)中的

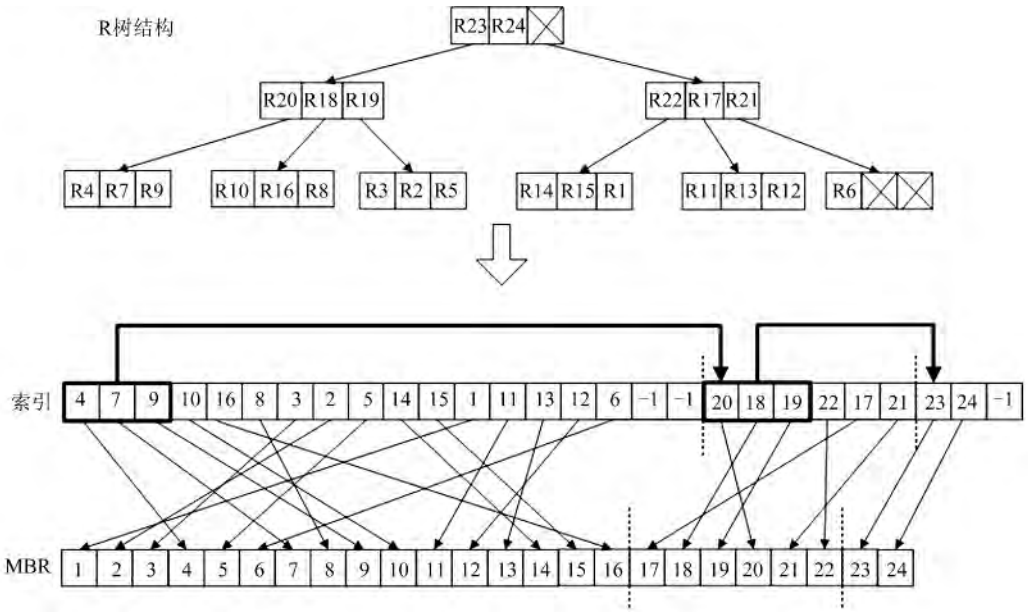

图1 基于数组的线性R树数据结构

Fig. 1 Data Structure of R-tree Based on Linear Array

排序算法。由文献[17]可知，当待排序的数组中数据类型为语言的原生类型时(如int、float等)，使用并行基数排序可以获得较高的性能。因此，在C-STR算法的步骤1)和步骤3)中，如果直接使用点结构体作为key进行并行排序，将大大影响算法的整体性能。C-STR算法的步骤3)中需要对S个切片中的MBR分别进行排序，每一次排序只能在一个内核函数中执行，而GPU对于同时并发的内核函数数量有着严格的限制，在计算能力为2.0的设备中，并发的内核函数必须不多于16个。因此，在步骤3)中，当数据量巨大时，对每个切片的并行排序远远不能满足算法的并发性需求。

为了提高算法的并发性，本文提出了一种基于GPU的高效并行R树构建算法(G-STR)。

输入：当前层中大小为N、元素为矩形(MBR)的数组，树节点容量M

输出：包含N个空间对象的R树索引

1) 创建一个大小为N的整型数组Index，并设置为顺序数组，即元素的值与其下标相等，$\{0,1,2,\cdots,N-1\}$，使用N个线程并行完成该赋值过程。

2) 使用N个线程，并行计算N个MBR的中心点，将x和y坐标分别存储于xCenter和yCenter两个数组中。

3) 使用xCenter数组中点的x坐标作为key，Index数组为value，进行并行基数排序。

4) 创建一个大小为N的数组slice，使用N个线程记录每个索引对应的MBR所属的切片号，即slice[Index$x[i]$] $= i/M$，$(i=0,1,2,\cdots,N-1)$。

5) 重新把Index置为顺序数组，Index$[i]=i$，$(i=0,1,2,\cdots,N-1)$，用yCenter为key，Index为value，进行并行基数排序。

6) 用Index为map，slice为value进行并行聚集操作，结果存放于一个新的整型数组xSortedSlice中，即xSortedSlice$[i]$ = slice[Index$[i]$]，$(i=0,1,2,\cdots\cdots,N-1)$。

7) 以$xSortedSlice$为key，Index为value，进行稳定的并行基数排序。

8) 使用$\lceil\sqrt{N/M}\rceil$个线程并行，将Index数组中每M个值索引的矩形从MBR数组中取出，合并形成新一层的MBR。

9) 将上一步得到的新MBR数组作为输入，递归完成每一层节点的构建，直至最顶层只有一个MBR，算法结束。

算法在MBR和Index独立存储的基础上，将其中心点的x和y分别存于两个数组中，这样，在对MBR进行按key排序时，key和value的数组元素分别是int和float型的，采用并行基数排序能大大提高效率。此外，在步骤5)和步骤7)中，对N个MBR用两次整体的排序(其中后一次是稳定排序)代替C-STR算法中对每个切片分别进行的按y坐标排序。具体做法为先确定每个MBR所属的切片号，继而对MBR及切片号按照中心点的y坐标进行排序，最后按切片号对MBR进行稳定排序，从而得到按切片号排列的

MBR。而且因为之前的 MBR 在 y 方向是有序的,并且最后一次排序是稳定排序,从而在每个切片内,MBR 保持了上一步在 y 方向上的有序性,这个改进大大提高了处理器的占有率,细化了并行的粒度。

3　实验与讨论

利用实验对本文方法的有效性进行验证,并通过运行时间与 CPU 上的 STR 方法进行效率上的对比,CPU 上的 STR 算法来自于 Geos 算法库。实验的硬件配置为:Intel® Core™ i5-2400 @ 3.10 GHz CPU,4 GB 主存,NVIDIA GeForce GTS 450 GPU;软件环境为 Windows7 64 位旗舰版,Visual Studio 2010,CUDA Toolkit 5.0。本文采用了 4 组要素数量由小到大的数据,为线与面两种较具代表性的数据(见表 1)。加速比是同一个任务在单处理器系统和并行处理器系统中运行消耗的时间的比率,在并行算法研究中被广泛用于衡量算法的性能,也是本文评估 G-STR 方法并行效率的重要指标。

表 1　实验数据

Tab.1　Experimental Data

名称	内容	类型	要素数量
Land-Use	江苏省土地利用数据	面	77 121
Base-Roads	道路基础数据	线	417 564
Detailed-Roads	1∶25 万道路数据	线	1 042 471
Pop-Hu	美国人口普查与居民地	面	11 078 297

分别使用 C-STR、G-STR 两种方法,对表 1 中 4 组实验数据进行 R 树空间索引结构创建实验,因为本文关注的是空间索引的创建以及查询,所有建树时间中未包含计算要素 MBR 的时间。此外,实验中基于GPU的建树及查询时间统计未包括数据在内存与显存之间的传输时间,所有的实验数据均为同条件下 10 次测试结果的均值。

本文从数据量的大小以及 R 树的节点容量两个方面来评估 R 树批量构建的效率,分别在 CPU 上使用 Geos 库中的 STR 方法和本文提出的 G-STR 算法,选用多组不同大小的矢量数据,进行 R 树的批量构建,并比较不同节点容量对建树时间的影响。从图 2 中可以看出,虽然 GPU 的单个计算核心的处理速度低于 CPU,但由于 G-STR 算法避免了自定义结构体低效率排序,并使用整体排序代替了分段排序,细粒度的并行算法充分利用了 GPU 的硬件资源,因此效率大大高于 CPU 算法。从图 2 中还可以看出,当 R 树节点容量低于 16 时,会引起效率的降低,而当节点容量达到 64 以上时,对建树效率的影响在逐渐减小。这是因为 R 树的节点容量的减小会引起树的高度增加,而 STR 方法在构建 R 树时按层构建,不利于并行粒度的细化,尤其在接近根节点的层次,由于节点数量较少,并发数较低,不能充分利用并行资源。

图 3 比较了 R 树的节点容量为 64 时,使用不同大小的数据建树,要素数量对建树性能的影响。要素个数增加时,数据的密集性大大提高,而本文提出的 G-STR 算法实现了数据级的并行,当数据规模变大时,硬件资源的利用率也相对提高,因此时间开销并没有像 CPU 上算法那样快速上升。表 2 列出了 G-STR 算法相对于 C-STR 算法的加速比,从测试结果可以看出,加速比随着 R 树节点容量的增大以及要素数量增加,都呈现出了变大的趋势。加速比随着数据规模的增加而变大的趋势则表明,G-STR 算法在面对海量空间数据时具有明显的优越性。

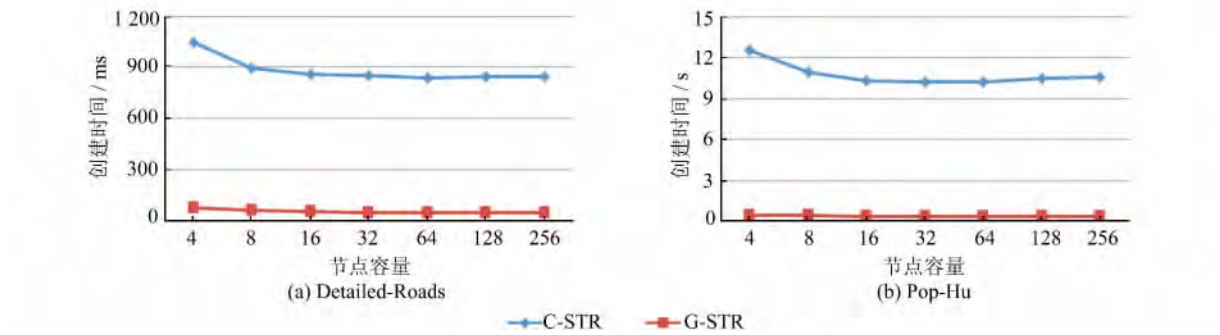

图 2　不同数据的建树时间对比

Fig.2　Comparing of Construction Time Using Different Data

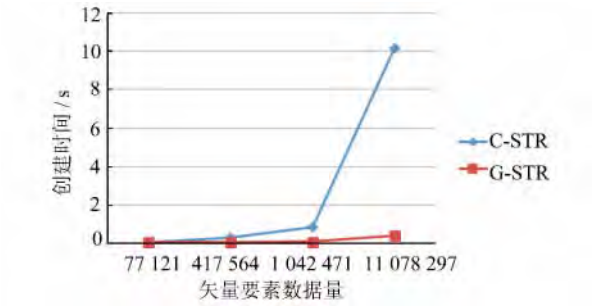

图3 节点容量为64时数据量与建树时间的关系

Fig. 3 Relationship Between Data Size and Construction Time with Node Capacity 64

表2 G-STR算法的加速比

Tab. 2 Speedup of G-STR Algorithm

节点容量	数据			
	Land-Use	Base-Roads	Detailed-Roads	Pop-Hu
4	2.10	7.29	13.71	26.67
8	2.42	7.73	13.93	26.46
16	3.05	8.75	15.35	26.06
32	3.40	9.03	16.39	25.34
64	3.81	9.30	16.01	25.87
128	3.72	10.19	16.80	26.81
256	3.74	10.54	17.23	27.31

4 结 语

本文结合GPU的可并行性与STR方法构建R树的高效性，提出了一种GPU上使用STR方法高效构建R树的算法，对比相应的CPU算法，取得了较高的加速比。由于GPU的众核特点，其更适合细粒度的并行计算，虽然空间索引具有可并行性，但对串行算法的直接并行化并不能在GPU上取得很好的效果。为提高并行粒度，提高内存访问效率，需要重新设计索引结构的数据组织以及并行算法，本文采用了两个独立的线性数组来描述一棵R树，并使用整体排序的策略提高STR方法中计算切片的并行度。本文通过实验讨论了数据量与算法加速比之间的关系，本文算法在实验中最高可以获得27倍的加速比。使用GPU上的R树查询对上述算法所创建的R树索引进行验证，该索引既可以用作持久化存储，也可以在GPU上即时实施查询操作。本文研究表明，在GPU环境中，使用细粒度并行算法可以大大提高R树索引的效率，在CPU频率难以进一步提高的情况下，为空间索引性能进步提供了新的途径。

由于STR方法中仅考虑了要素MBR的中心点，如果结合要素的大小或形状的聚类方法将能有效提高R树的质量，而这类方法往往带来更大的计算量，使用基于GPU的细粒度并行方法将有望实现快速且高质量的R树构建。此外，为适应云环境下的空间数据管理，使用多GPU的分布式并行R树索引将是下一步工作的重点。

致谢：感谢地球系统共享平台——长三角平台(http://nnu.geodata.cn)提供数据。

参 考 文 献

[1] Papadopoulos A, Manolopoulos Y. Parallel Bulk-loading of Spatial Data[J]. *Parallel Computing*, 2003, 29(10): 1 419-1 444

[2] Guttman A. R-trees: A Aynamic Index Structure for Spatial Searching[C]. ACM SIGMOD, Boston, MA, 1984

[3] Sellis T, Roussopoulos N, Faloutsos C. The R+-tree: A Dynamic Index for Multi-dimensional Objects[C]. The 13th VLDB, Brighton, England, 1987

[4] Bechmann N, Kriegel H P, Schneider R, et al. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles[C]. SIGM OD, Atlantic City, New Jersey, 1990

[5] Roussopoulos N, Leifker D. Direct Spatial Search on Pictorial Databases Using Packed R-trees[C]. ACM SIGMOD, Austin, TX, 1985

[6] Liu J, Liu T, Wu H, et al. From Graphic Processing Unit to General Purpose Graphic Processing Unit[J]. *Journal of Wuhan University (Natural Science Edition)*, 2013, 59(2): 198-206(刘金硕，刘天晓，吴慧，等. 从图形处理器到基于GPU的通用计算[J]. 武汉大学学报(理学版), 2013,(2): 198-206)

[7] Simion B, Ray S, Brown A D. Speeding up Spatial Database Query Execution Using GPUs[J]. *Procedia Computer Science*, 2012,(9): 1 870-1 879

[8] Kim J, Hong S, Nam B. A Performance Study of Traversing Spatial Indexing Structures in Parallel on GPU[C]. IEEE 14th International Conference on High Performance Computing and Communications. Liverpool, United Kingdom, 2012

[9] Yang K, He B, Fang R, et al. In-memory Grid Files on Graphics Processors[C]. The 3rd International Workshop on Data management on New Hardware, ACM, Beijing, China, 2007

[10] Lijuan L, Wong M D F, Leong L. Parallel Implementation of R-trees on the GPU[C]. The 17th Asia and South Pacific Design Automation Conference, San Francisco, USA, 2012

[11] You S, Zhang J. GPU-based Spatial Indexing and

Query Processing Using R-Trees[EB/OL]. http://www-cs. ccny. cuny. edu/~jzhang/papers/rtree_tr. pdf, 2012

[12] 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

[13] Kamel I, Faloutsos C. On Packing R-trees [C]. CIKM, Washington D C, USA, 1993

[14] Zhang Mingbo, Lu Feng, Shen Paiwei, et al. Evolvement and Progress of R-tree Family[J]. *Chinese Journal of Computers*, 2005, 28(3): 289-300 (**张明波,陆锋,申排伟,等. R 树家族的演变和发展**[J]. **计算机学报**, 2005, 28(3): 289-300)

[15] Garcia Y, Lopez M, Leutenegger S. A Greedy Algorithm for Bulk Loading R-trees [C]. The 6th ACM-GIS, Washington D C, 1998

[16] Lee T, Lee S. Omt: Overlap Minimizing Top-down Bulk Loading Algorithm for R-tree[J]. *Advanced Information Systems Engineering*, 2003(7): 69-72

[17] Micikevicius P. Comparison-Based In-place Sorting with CUDA[M]//Wen-Mei W H. GPU Computing Gems. Waltham, MA: Morgan Kaufmann, 2011

GPU-Based Parallel Bulk Loading R-trees Using STR Method on Fine-Grained Model

SHAO Hua[1]　*JIANG Nan*[1]　*HU Bin*[1]　*LV Heng*[1]　*ZHU Jin*[1]

1　Key Laboratory of VGE, Ministry of Education, Nanjing Normal University, Nanjing 210023, China

Abstract: In the era of big data, efficient spatial indexes need to be established quickly for massive spatial data. The R-tree spatial index built by the sort tile recursive (STR) technique has excellent query performance but low efficiency when building. We propose an R-tree bulk loading algorithm using a STR technique based on general purpose computing on a GPU. A linear array structure is used to store an R-tree and an overall sorting algorithm is used instead of segmented sorting. Experiments show that our proposed algorithm achieves up to a 27 speedup. Our experiments also indicate that the speedup increases as the data becomes larger. We use a query algorithm on the GPU to verify the R-tree bulk loading algorithm; finding that it has good query performance. Our algorithm takes advantage of the parallel processing capacity of the GPU and achieves high efficiency which shows that the technology of GPU computing has broad applicability in the spatial indexing field.

Key words: R-tree; GPU; bulk lading; fine-grained parallel; spatial index

First author: SHAO Hua, PhD candidate, specializes in high performance spatial analysis and spatial data mining. E-mail: shyxiaoxiao@163. com

Corresponding author: JIANG Nan, professor. E-mail: njiang@njnu. edu. cn

Foundation support: The National Key Technology R&D Program of China, No. 2012BAH35B000; the National R&D Infrastructure and Facility Development Program of China; the Priority Academic Program Development of Jiangsu Higher Education Institutions.