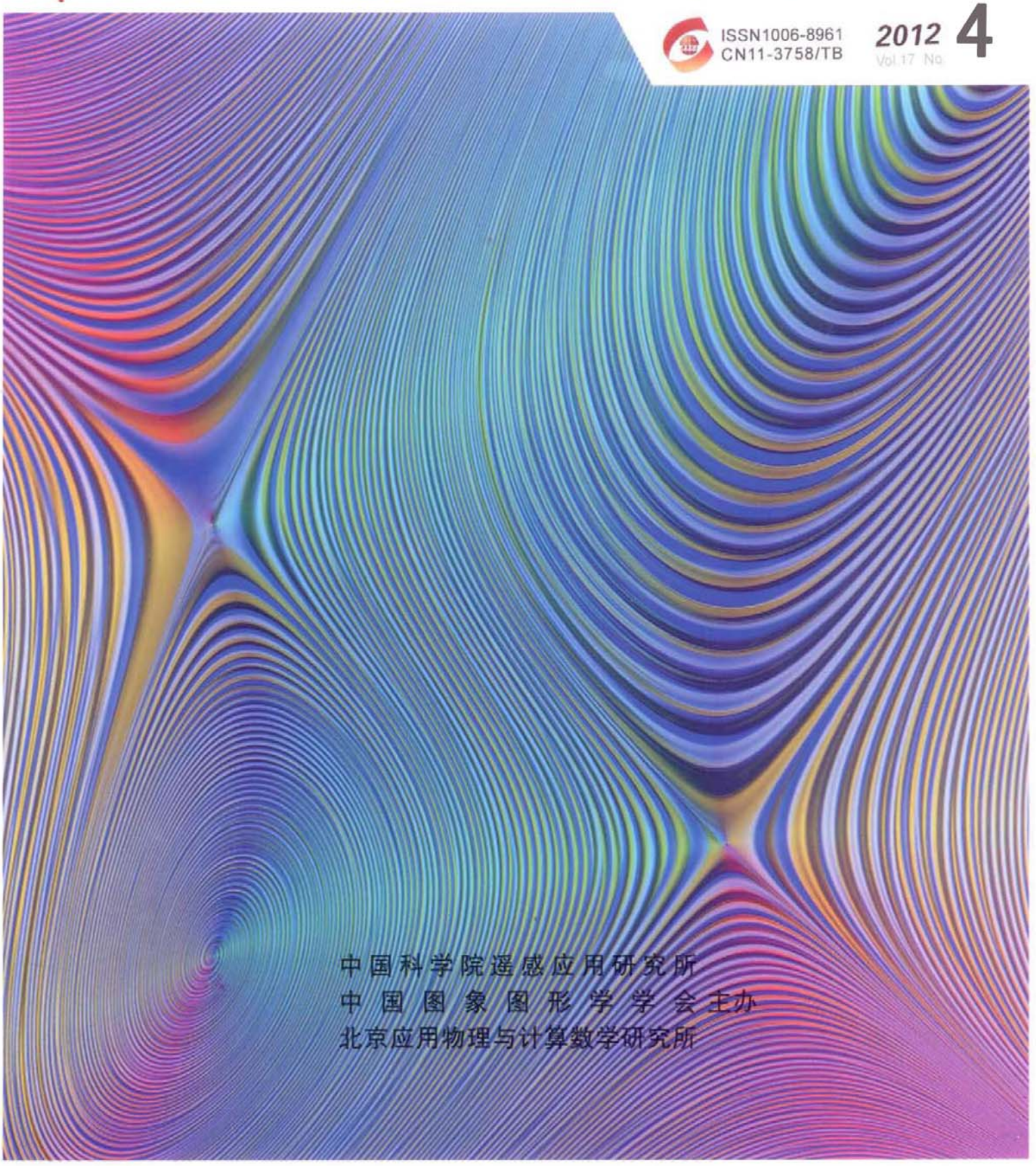
Journal of Image and Graphics

中国图象图形学报

ISSN1006-8961
CN11-3758/TB
2012 Vol.17 No. 4



中国科学院遥感应用研究所
中国图象图形学学会主办
北京应用物理与计算数学研究所

中国精品科技期刊

中国图象图形学报

Zhongguo Tuxiang Tuxing Xuebao

2012 年 4 月　第 17 卷　第 4 期(总第 192 期)

目　次

© 中国图象图形学报版权所有

中国图象图形学报

刊名题字:宋 健　　月刊(1996 年创刊)　　第 17 卷 第 4 期　　2012 年 4 月 16 日出版

主管单位 中国科学院
主　　办 中国科学院遥感应用研究所
中国图象图形学学会
北京应用物理与计算数学研究所
主　　编 李小文
编辑出版 《中国图象图形学报》编辑出版委员会
北京 9718 信箱　邮编 100101
电子信箱:jig@ irsa. ac. cn
电话:010-68407995　010-82614429
网　址:www. cjig. cn
印刷装订 北京北林印刷厂
广告经营许可证 京朝工商广字第 0346 号
总 发 行 北京报刊发行局
订　　购 全国各地邮局
国外发行 中国国际图书贸易总公司
(中国国际书店)
(北京 399 信箱　邮编 100044)

Superintended by Chinese Academy of Sciences
Sponsored by Institute of Remote Sensing Application, CAS China Society of Image and Graphics
Institute of Applied Physics and Computational Mathematics
Chief editor LI Xiaowen
Editor, Publisher Editorial and Publishing Board of Journal of Image and Graphics
(P. O. Box 9718, Beijing 100101, China)
E-mail: jig@ irsa. ac. cn
Distributed by Beijing Bureau for Distribution of Newspapers and Journals
Domestic All Local Post Offices in China
Foreign China International Book Trading Corporation
(P. O. Box 399, Beijing 100044, China)
Printed by Beijing Beilin Printing House

ISSN 1006-8961　CN11-3758/TB　CODE ZTTXFZ　国内邮发代号:82-831　国外发行代号:M1406　国内定价:45.00 元

© 中国图象图形学报版权所有

第 18 届中国遥感大会征文通知

“第 18 届中国遥感大会”将于 2012 年 10 月 19 日 - 23 日在武汉召开。本届会议由中国遥感委员会主办,中国测绘学会摄影测量与遥感专业委员会和武汉大学承办。会议将围绕“遥感—全方位的社会服务”这一宗旨,以遥感学界院士与知名专家的特邀报告,分会场专题技术交流与技术讲座,重点项目研讨汇报、技术展览,新技术与新产品发布,专业委员会理事会等多种形式开展,同时举行“第 7 届中国青年遥感辩论会”和“第 2 届全国高分辨率遥感数据处理与应用研讨会”。

会议将全方位地展示遥感(RS)、全球定位系统(GPS)、地理信息系统(GIS)等方面的最新成果,为专家、学者和政府主管部门搭建联系纽带,为研发和用户提供技术交流平台,共同促进遥感科技的发展、遥感产业化的推进和大遥感体系的建立。

本届会议围绕大会主题将就遥感新理论、技术、方法和应用进行征文,范围包含但不限于以下方面:

1)国家遥感中长期发展战略、国际遥感前沿与进展;

2)航天、航空、低空、地面遥感技术及系统;

3)光学、红外、高光谱及激光遥感技术;

4)主、被动微波及雷达遥感技术;

5)数字摄影测量与制图;

6)高分辨率遥感数据处理与应用;

7)地理空间数据处理技术与方法;

8)地理国情监测(土地、农业、林业、矿产、环境、地质及水资源等);

9)海洋、气象与全球变化;

10)遥感、地理信息系统与导航定位系统(3S)集成与应用;

11)智慧城市与数字地球;

12)深空探测与行星测绘;

13)教育、培训与社会公共事业。

征文采用在线方式投稿;

投稿要求:论文内容不涉密,且未在国内外学术刊物或正式学术会议上发表过;被录用的全文将收入大会论文集(送 ISTP 检索),并精选 70 ~ 90 篇口头报告论文编辑出版英文 SPIE 会议文集;大会将评选青年优秀论文(参加口头报告),论文将直接进入英文 SPIE 会议文集。

论文摘要截止日期为 2012 年 5 月 15 日,全文截稿日期为 2012 年 6 月 15 日。

会议相关信息,请查阅会议网址:http://rsgis.whu.edu.cn/18ccrs/index.html

“第 18 届中国遥感大会”组委会

第 33 届亚洲遥感会议征文通知

“第 33 届亚洲遥感会议”将由泰国地理信息和空间技术发展局(GISTDA)、科技部(MOST)和亚洲遥感协会(AARS)联合主办,于 2012 年 11 月 26 - 30 日,在泰国芭堤雅市宗滴恩酒店举行。这是亚洲遥感协会每年一届的系列学术会议。本届大会征文包括传感器与平台、算法和图像处理、GIS 与 Web GIS、全球导航卫星系统、灾害、自然资源、环境科学、教育和宣传、健康科学、制图、其他等方面。

会议重要日期:

论文摘要提交截止:2012 年 5 月 15 日;

论文接收通知:2012 年 7 月 1 日;

论文全文提交截止:2012 年 9 月 30 日;

网上注册截止:2012 年 10 月 26 日;

会议召开日期:2012 年 11 月 26 - 30 日。

会议还将组织学生专场和技术展览,其他信息请访问会议网站:http://acrs2012.gistda.or.th

与往年一样,中国遥感委员会仍将鼓励中国遥感科研人员和企事业单位参加会议,并组团参加学术交流和会议展览。

中国遥感委员会

© 中国图象图形学报版权所有

Journal of Image and Graphics

(Monthly, Started in 1996)

Vol. 17 No. 4 April 2012

Contents

© 中国图象图形学报版权所有

中图法分类号: TP301.6 **文献标志码**: A **文章编号**: 1006-8961(2012)04-0575-07
论文引用格式:赵艳伟,程振林,董慧,方金云. 图形处理器空间插值并行算法的实现[J]. 中国图象图形学报,2012,17(4):575-581

图形处理器空间插值并行算法的实现

赵艳伟[1,2],程振林[1],董慧[1,2],方金云[1]
1. 中国科学院计算技术研究所,北京 100190; 2. 中国科学院研究生院,北京 100049

摘 要: 空间插值是地理信息系统(GIS)空间分析中计算复杂且耗时的操作,因此无法满足实时性的要求。随着图形处理器(GPU)浮点计算能力的大幅提高,GPU 通用计算已成为处理 GIS 领域内复杂计算的研究热点。为实时化一些传统低效的算法提供了良好的契机。利用 GPU 在并行计算上的优势,将反距离加权法插值算法映射到了统一计算设备架构(CUDA)并行编程架构。首先在 GPU 中建立二级索引使计算层次得到了合理的划分,然后利用多线程分块策略执行并行插值计算。最后通过实验表明,该方法的插值误差与 CPU 方法相比能控制在 10^{-6} 数量级,并且在插值半径较大插值数据较多的情况下,该算法可达到 40 倍以上的加速比。充分证明了该方法的正确性及高效性。

关键词:地理信息系统;并行插值;图形处理器; 统一计算设备架构

Realization of GPU parallel spatial interpolation method

Zhao Yanwei[1,2], Cheng Zhenlin[1], Dong Hui[1,2], Fang Jinyun[1]
1. *Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100190, *China*;
2. *Graduate University of Chinese Academy of Sciences, Beijing* 100049, *China*

Abstract: Interpolation is one computational complex and time-consuming operation in the fields of spatial analysis that can not meet the real time demand. With the rapid increase of GPU floating-point computing power, general-purpose computation on graphics processors (GPGPU) has became an evolving research field in spatial information processing, and it provides an opportunity to accelerate some traditional inefficient algorithms. In this paper, we map the inverse distance weighted (IDW) interpolation method to the compute unified device architecture (CUDA) parallel programming model. Taking the advantage of graphics processing unit (GPU) parallel computing, we build two-level indexes on GPU, then blocking schemes are used to assign computing task among different threads. After illustrating the parallel interpolation process, we conduct several experiments, The experiment result shows that the error of this new method can control under 10^{-6} compared with CPU-based method. With larger influence radius and massive data, the performance can obtain above 40 times speedups over a very similar single-threaded CPU implementation. It is demonstrated the correctness and high efficiency of our optimized implementation.

Key words: geographic information system; parallel interpolation; graphics processing unit; compute unified device architecture

收稿日期:2011-06-18;**修回日期**:2011-10-08
基金项目:国家高技术研究发展计划(863)基金项目(2009AA12Z220,2009AA12Z226)
第一作者简介:赵艳伟(1985—),女,满族,中国科学院计算技术研究所计算机应用专业博士研究生,主要研究方向为 WebGIS 空间分析,并行计算等。E-mail: zhaoyanwei@ict.ac.cn

© 中国图象图形学报版权所有

0 引 言

随着地理信息领域的发展,人们面临的信息处理问题越来越复杂,空间数据规模越来越庞大。空间插值作为GIS(地理信息系统)领域空间分析的一项重要技术,是指在给定一系列离散样本点的基础上,根据其特征值与空间位置拟合出一个函数方程。该函数能通过已采样点的特征值来推算未采样点值,从而在一定区域范围内形成连续特征表面。

目前,国内外有许多学者对空间插值进行了大量的研究,但大部分关注的交点都集中于提高插值精度等问题上[1-3]。由于插值过程的计算量大、计算密度较高、浮点运算复杂,导致了其缺点是计算速度慢,并且执行效率将随着采样点的增加而急剧降低。对于一些要求实时性很高的交互式应用场景,运算速度将是用户更为关心的一个因素。一个精度适当、响应迅速的插值结果往往比一个精度更高但响应时间较长的结果更有意义。因此为适应这种新型应用,设计一个高效的插值算法是十分必要的。

传统提高性能的主要手段之一是提高处理器的时钟频率,然而由于受工艺、材料和功耗等物理限制,人们已无法在现有的架构上通过该手段得到满意的性能提升。因此,并行计算逐渐成为GIS领域发展的新趋势,中央处理器也不断向多核的方向发展[4],双核图形处理器(CUP),4核CPU已经广泛应用到了多核空间信息处理中。然而这种基于多个CPU的并行计算对于线程启动、切换、通信等操作的开销是很大的,并且对于一些高强度计算问题,由于受计算能力和计算资源等限制,其性能提升的空间也是有限的[5]。近几年来,图形处理器以其众多的流多处理器(stream multi-processor)而越来越受到人们关注,众核框架为进一步提高并行计算效率提供了发展空间。CUDA[6]作为一种新的统一计算设备架构,使GPU在通用计算领域中的可编程性不断提高,目前国内外已经有很多采用CUDA架构实现并行计算的成功案例,得到了几倍、几十倍甚至上百倍的加速比。相应的文章也发表在各大主流的杂志及会议上。这些文章涉及的领域十分广泛,如几何计算[7]、碰撞冲突检测[8]、数据库系统[9]、流体动力学[10]、图像处理[11]、地理信息系统[12]等领域。虽然领域不同,但凡是得到较高加速比的应用均具有海量数据、计算强度大、浮点运算复杂等特点。

基于CUDA的GPU并行计算已成为GIS领域的研究热点。本文以插值方法中最常用的反距离加权法(IDW)[13]为例,设计了一种基于CUDA的快速插值并行算法,旨在说明如何利用GPU强大的计算能力,通过CUDA并行计算模型加速GIS中计算密集型算法,从而为地理信息领域并行计算的研究与实践提供一个创新性的尝试和良好的解决方案。本文提出并详细地描述了该并行插值算法的实现细节,通过可视化对比与效率对比等实验,表明该算法在可行性、正确性及并行效率方面均具有良好的性能。

1 CUDA计算模型

CUDA是NVIDIA公司推出的统一计算设备架构,它能够使开发人员不必过多关注显卡细节,而通过类C语言开发GPU上可执行的并行算法。NVIDIA GeForce 9800 GTX+的硬件结构包含了16个流多处理器(SM),每个SM又包含8个标量流处理器(SP)做为算数逻辑单元(ALU),它们共用同一套取址与发射单元。每个SM共享16KB的共享内存,以便于SM内的线程互相通信。

CUDA以线程为单位执行运算,这些线程在一个线程网格(grid)中被组织成两个层次:粗粒度层次中,一个线程网格包含多个线程块(block);细粒度层次中,每个线程块包含数量相等的线程。通过内置变量,可以方便计算block的索引号和线程(thread)的索引号。

在CUDA模型中,kernel是在设备上执行数据并行的核心函数,当一个任务被分配到GPU中,线程将唤醒kernel函数。一个典型的CUDA计算包括以下几个步骤:在主机端注册设备;申请显存空间并将要并行处理的数据拷贝到设备的全局内存中;启动kernel函数执行并行任务;将计算结果拷贝回内存;释放显存空间。

2 并行空间插值算法实现

由于该算法的输入为一系列离散的点,因此为了快速检索在目标插值点确定范围内的所有采样点,首先需要建立一种在GPU中使用的新型索引,然后利用CUDA的2维线程及纹理机制,对目标点进行分块,使每个线程映射到一个目标点进行插值计算,从而使并行达到最大化。

算法流程:1)将所有离散采样点存入GPU的1

© 中国图象图形学报版权所有

维纹理存储器中;2)对输入数据在显存中建立全局瓦片索引;3)对相对独立的线程块将其对应的瓦片索引拷贝到共享内存并建立相应的位检索码;4)根据线程任务划分,启动 kernel 设备函数执行插值运算,保证线程块中的每个线程计算一个目标点;5)将插值结果从显存传回主存。

2.1　建立索引

在 GPU 中,数据计算是按照线程块(block)来划分的,对于所有线程块之间的通信可以通过访问显存(global memory)实现,但读写显存的延迟很高;然而在每个特定的线程块内部,线程访问该 block 的共享内存的速度很快,所以应该最大程度地使用共享内存。基于以上 GPU 存储器的特点,建立了一种瓦片网格索引(*TGI*)。瓦片结构是指,将插值区域按照插值精度及指定格网大小建立线性结构。其过程分为全局计算和分块计算两个层次。全局计算是指根据输入的采样点先在显存中建立一个统一的外部哈希表,分块计算是在全局哈希表的基础上取出相应的部分拷贝到共享内存,并建立其对应的位检索码。

1)全局计算步骤

首先计算原始输入点集的区域范围。设单元格网的分辨率为 ω,插值受影响的半径为 R,根据 CUDA SDK 中的并行规约算法计算所有输入数据集的外包,设为 *Bbox*,则输入点集转换为格网矩阵 *input*(*ori_width*,*ori_height*)的大小即为该 *Bbox* 的范围。

其次根据输入点集的范围进一步确定全局 *TGI* 索引的范围。设每个瓦片的 *TGI* 由 $S \times S$ 个单位格网构成,则原始矩阵 $\boldsymbol{\Omega}$ 可被划分为 *TGInum_row* × *TGInum_col* 个瓦片网格索引,行列参数可按如下公式计算:

$$TGInum\_row = \left| \frac{ori\_width}{\omega S} \right| \tag{1}$$

$$TGInum\_col = \left| \frac{ori\_height}{\omega S} \right| \tag{2}$$

对于 *TGI* 中的任意一个瓦片 $TGI(i,j)$ 与原始矩阵 $\boldsymbol{\Omega}$ 内部数据的对应关系为

$$TGI(i,j) = \bigcup_{(a,b) \in [0,S-1]^2} \boldsymbol{\Omega}[a + iS, b + jS] \tag{3}$$

如图 1 所示,当输入矩阵的大小无法被划分为整数个瓦片时,我们需要将瓦片外扩,以覆盖输入区域范围。

最后对所有输入点 $p \in P$ 根据其 (x,y) 坐标按照如下的哈希公式计算:

$$hash(p_i) = \left| \frac{p_{i.}y}{\omega S} \right| \times \left| \frac{ori\_width}{\omega S} \right| + \left| \frac{p_{i.}x}{\omega S} \right| \tag{4}$$

$TGInum\_row \times TGInum\_col \times S^2$ 为该哈希表的长度,如此能保证每一个输入点 i 都对应一个唯一的外部键值(ex_key_i),而一个外部键值则对应多个输入点并且数目小于 S^2,具有相同外部键值的坐标保证落入同一个瓦片索引中。

与此类似,外部键值确定后要对落入相同瓦片的输入点通过哈希公式(5)计算其内部键值,并用一个值域变量(*value*)记录该采样点的索引,即 $value_i = i$。

$$hash'(p_i) = p_{i.}y\%(\omega S) + p_{i.}x\%(\omega S) \tag{5}$$

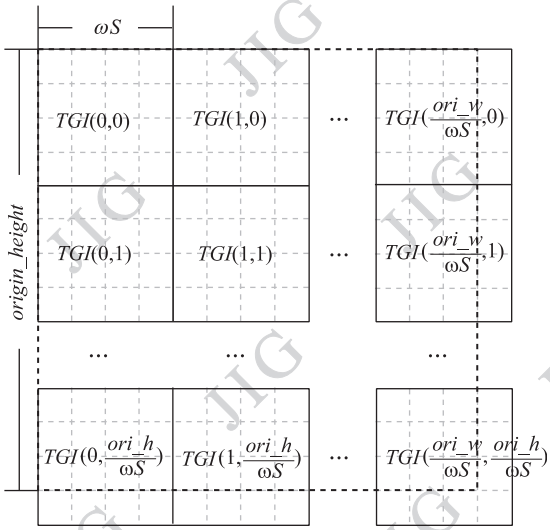


图 1　*TGI* 划分规则($S=4$ 情况)

Fig. 1　The division rules of *TGI* size ($S=4$)

图 2 描述了外部键值与内部键值的关系,也就是说对于任意一个输入点既要计算它隶属于哪个瓦片又要计算它在当前瓦片中的偏移位置,最后根据每个输入点的上述 3 个变量,将(ex_key_i,$value_i$)构成一组 *key-value* 对,利用 CUDPP 的 cudppSort 并行

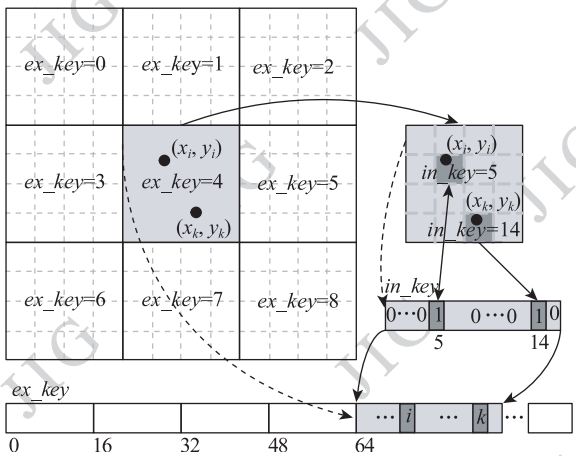


图 2　外部键值及内部键值的对应关系

Fig. 2　The relationship between external key and internal key

© 中国图象图形学报版权所有

算法进行排序,这样就保证了所有输入点按照外部键值由小到大连续的存储在全局显存中,其中具有相同外部键值的点根据内部键值的哈希表实现了局部连续性。这种做法的好处是避免了传统索引结构的指针机制,通过线性连续存储结构,满足了 CUDA 的内存管理的模式,即读取连续的显存地址,进而可以采用内存融合的模式提高访问速度。

2)分块计算步骤

上述步骤实现了在全局存储器中对所有输入点构造了全局索引。由于 GPU 中的线程对显存(global memory)的访问每次要占用上百个时钟周期,在执行插值计算时如果所有线程都通过全局索引查找采样点必将影响查找速度,因此根据 GPU 中特有的线程块及共享内存(shared memory)等机制在此基础上进行优化,建立二层分块索引(Tile Block Index,*TBI*)。该部分的主要目标是根据插值计算任务的分配情况,计算出应该映射到每个线程块共享内存中的全局索引。

如图 3 所示,对任一线程块,为了保证该块内的所有目标点在插值半径覆盖内的采样点都能在 *TBI* 中检索到,需要对每个 block 进行边界向四周扩充 *R*,然后利用 CUDA 中内嵌变量求出新区域的最小坐标和最大坐标,分别为:

$(BlockDim.x \times BlockIdx.x - R, BlockDim.y \times BlockIdx.y - R)$

$(BlockDim.x \times BlockIdx.x + R, BlockDim.y \times BlockIdx.y + R)$

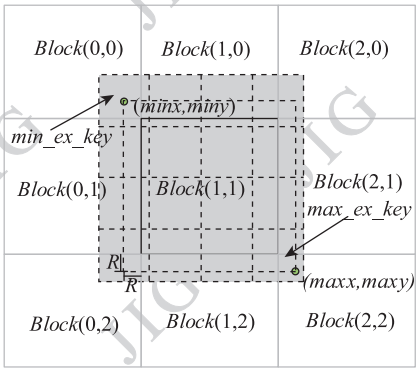


图 3 *TBI* 与 *TGI* 对应关系

Fig. 3 The relationship between *TBI* and *TGI*

接着按照上述计算外部键值的方法计算扩展后的区域覆盖的最小编号 *TGI* 和最大编号 *TGI*,分别设为 *min_ex_key* 和 *max_ex_key*,则当前 block 的 shared memory 中的索引 *TBI* 与 *TGI* 存在如下关系:

$$TBI_{i,j} = \bigcup_{\substack{a\in[0,\phi]\\ b\in[0,\eta]}} TGI\left[min\_ex\_key + a \times \left\lceil \frac{ori\_width}{\omega S} \right\rceil + b\right] \quad (6)$$

式中,

$$\phi = \frac{max\_ex\_key}{TGInum\_col} - \frac{min\_ex\_key}{TGI\ num\_col}$$

$\eta = \mathrm{mod}(max\_ex - key, \mathrm{TGInum\_row}) - \mathrm{mod}(\mathrm{min\_ex\_key}, \mathrm{TGInum\_row})$。最后,将每个 *block* 对应的 *TGI* 从全局存储器中拷贝到共享存储器中,同时建立相应的位检索码(*TBI* 中有值位用 1 表示,无值位用 0 表示),这样建立起来的索引保证了每个 *block* 在执行插值计算的时候只需访问自己内部的 *shared memory*,使相邻 *block* 之间的插值过程彼此独立、并且索引无冲突。

2.2 插值算法

按照算法流程,我们需要利用 *NVIDIA GeForce* 9 系列的 1 维纹理存储器存储离散采样点,对于该纹理中的每一个点将是一个 *float*3 类型的数据,其中 *x* 分量和 *y* 分量代表采样点的空间位置,*z* 分量代表特征值。

设执行插值过程的线程块组织形式为 $dst\_BS(\lambda,\lambda)$,其中既要保证线程总数 $\lambda^2 \leqslant 512$,又要使线程数量尽量为 half-warp 的整数倍,此处我们选择 $\lambda = 16$,即每个线程块有 256 个线程。若分派一个线程去完成一个对应位置目标点的插值计算,则 Grid 中需要的 block 数目可表示为

$$GridDim.x = \frac{ori\_width + dst\_BS.x - 1}{dst\_BS.x} \quad (7)$$

$$GridDim.y = \frac{ori\_height + dst\_BS.y - 1}{dst\_BS.y} \quad (8)$$

当 GPU 中数据准备好及索引建立完毕后,就可以按照线程任务划分的策略触发 kernel 设备函数,执行并行插值操作。算法处理过程中目标插值点可以根据 CUDA 获取线程编号的 2 维内嵌变量得到,使每个线程处理一个插值点。对于任一线程,则根据其空间位置求出在插值影响半径范围内的 TBI 集合,并利用相应的位移检索码进行位运算求出采样点的位置,根据偏移在 shared memory 中检索该位置的索引,进而通过 tex1D 取得采样点特征值,执行插值计算。伪代码如下:

算法 1 __global__ IDWkernel()

$x \leftarrow$ __mul24(*blockDim.x*, *blockIdx.x*) + *threadIdx.x*;

$y \leftarrow$ __mul24(*blockDim.y*, *blockIdx.y*) + *threadIdx.y*;

float *sum*1, *sum*2;

© 中国图象图形学报版权所有

```
bound(W) ←( x-R,x+R,y-R,y+R);
ξ←load_shared_TBI(bound(W));
foreach ε in ξ
    Ð←idx_position(ε);
    foreach φ in Ð
        float3 smp←tex1D(tex, φ);
        d←sqrt(smp.x^2 + smp.y^2);
        if d≤R then
            sum1 + = smp.z/(d*d);
            sum2 + =1/(d*d);
        end if
    end for
end for
output[y][x].z← sum1/sum2;
```

其中位检索码是一种与 *TBI* 相对应的线性结构,对 *TBI* 中有值的点,位检索码记为 1,否则记为 0。若瓦片的 $S=4$,则每个瓦片的位检索码为 16 位由 0 和 1 组成的二进制数。其工作方式通过不断地迭代每个瓦片的二进制数求出该瓦片内值为 1 的位置集合。这样做的好处是该查找过程是基于位运算的,因此对较稀疏的数据可以快速地定位其值所在的位置。伪代码如下:

```
算法 2   __device__   idx_position()
Ð ← Φ
temp←value
do
    temp←value & (value-1);
    k←value - temp;
    Ð + =k;
    value←temp;
while(temp)
return Ð ;
```

将算法 1 与传统 CPU 实现对比可知,通过 GPU 实现的并行插值算法较串行方法减少了迭代次数,原来需要顺序地对每个目标点进行插值计算,经过 CUDA 并行,所有目标点可以同时执行各自的插值,而且互不影响,极大的提高了效率。但同时观察到,并行过程中要用到全局显存和共享内存建立索引,虽然浪费了一定的存储器资源并且建立索引也要花费一定的时间,但对于大规模密集数据的复杂算数运算,性能提升的幅度可以弥补建立索引的时间损失,因此是值得的。

3　实验分析

实验部分主要关心该并行插值算法的两方面指标。第一方面是插值效果,对于利用不同的硬件环境 GPU 实现的插值算法,首先在插值效果上应尽量和 CPU 实现的方法保持一致。第二方面是插值的效率,这部分主要对比与 CPU 方法在性能方面的差异。

3.1　实验环境

CPU 部分:Intel® Core™ 2 Duo CPU,3.18 GHz,2 GB系统内存,Microsoft Windows XP Pro SP2 操作系统,Microsoft Visual Studio 2005 开发环境。GPU 部分:GeForce 9800 GTX + 显卡,512MB 显存,128 个流处理器,NVIDIA driver 2.2,CUDA sdk 2.2,CUDA toolkit 2.2。

3.2　可视化对比

该部分实验的采样点集合由 Matlab 模拟生成的格网 DEM 高程矩阵组成,如图 4 所示,其中该采样 DEM 的 x、y 坐标对应采样点的空间位置,z 值作为采样点的高程值,为 10×10 矩阵,通过 Matlab 产生。

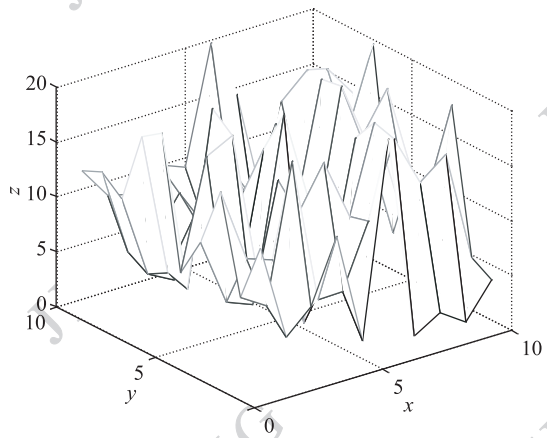


图 4　原始采样图

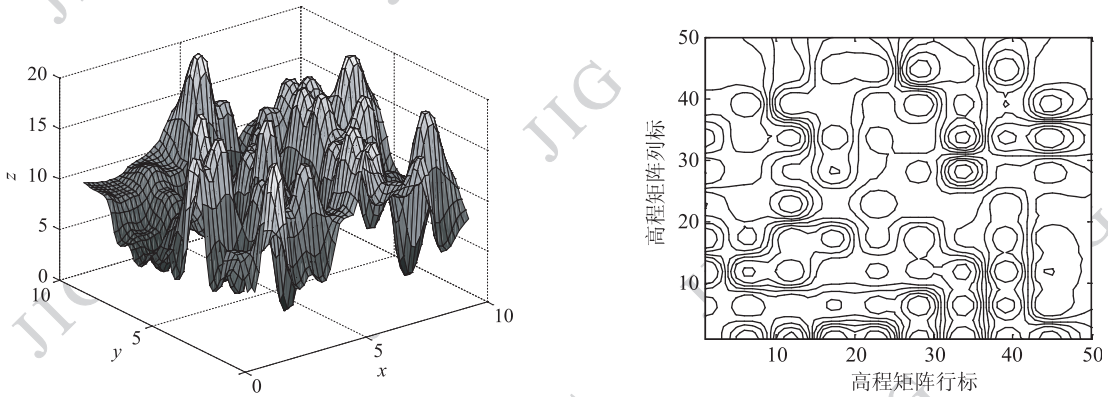
Fig. 4　Original samples

CPU 插值过程主要利用 MEX 技术实现 Matlab 与 C++的混合编程,利用衍生的 mex32 文件,插值计算执行 C++实现的 IDW 插值函数,插值结果为 50×50 矩阵,利用 Matlab 显示,如图 5(a)所示。其中图 5(a)是 DEM 插值结果的 3 维显示图,图 5(b)是该结果等价的等高线图。

GPU 插值过程的不同之处是 Matlab 要通过 nvmex 技术才能调用 CUDA 编写的在显卡上执行的 IDW 函数,完成插值后还是利用 Matlab 显示出来,如图 6 所示。

从图 5 与图 6 的对比可视化效果来看,GPU 的插值结果无异于 CPU 插值结果。图 7 为测得的上述实验结果的相对误差,可以看出插值结果的误差控制在 10^{-6}的数量级,产生这种误差的原因在于 GPU

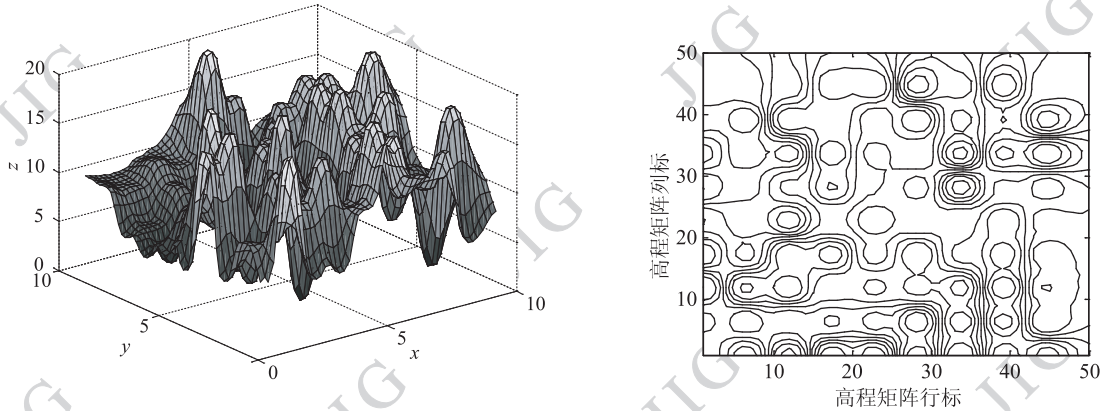
© 中国图象图形学报版权所有



(a) CPU插值结果3维显示图 (b) CPU插值结果等高线图

图 5 CPU 串行插值效果

Fig. 5 CPU interpolation result



(a) GPU插值结果3维显示图 (b) GPU插值结果等高线图

图 6 GPU 插值效果

Fig. 6 GPU interpolation result

中使用了设备代码支持的浮点内建函数,它们精度较低,但运算速度更快。对于某些实时性要求高的应用来说,牺牲较少的精度换取更高的效率提升是可以接受,并有实际应用价值的。

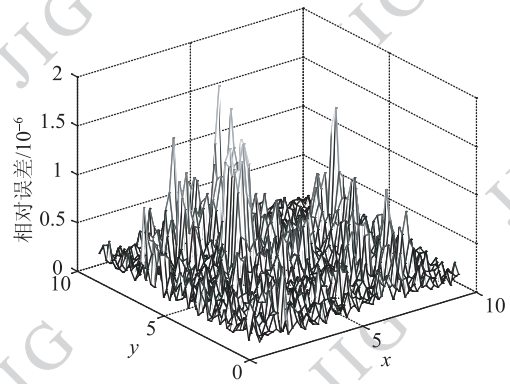


图 7 相对误差

Fig. 7 Relative error

3.3 效率对比

为了衡量 GPU 并行插值算法的性能,我们通过加速比这一指标来进行分析,加速比通常定义为

$$r_a = t_s / t_p$$

式中,t_s 为串行时间,t_p 为并行时间。

原始采样点为 2 000 组随机生成的(x,y,z)形式的离散点集,目标差值数据为多组随机生成的(x,y)点集,数据量在 10 000 ~ 25 000 之间。

首先需要考察的是插值权重因子对插值效率的影响,因此我们假设插值半径为 4,每个 block 内有 256 个活动线程,分别对 $u=2$ 和 $u=3$ 进行对比实验,由于权重因子越大意味着插值计算的幂次方越高,计算越复杂,从表 1 中的结果可以发现当权重因子较大时,整体插值的加速比是有一定提高的,虽然提高的幅度不是很明显,但也能够说明该算法对复杂算数运算的良好支持。

© 中国图象图形学报版权所有

其次进行的实验为测试不同插值半径对插值效率的影响,如图 8 所示。

表 1　权重因子对插值效果的影响

Table 1　The interpolation effect of different weight factor

数据集/10^4	GPU 时间/ms		CPU 时间/ms		GPU/CPU 加速比	
	$u=2$	$u=3$	$u=2$	$u=3$	$u=2$	$u=3$
1	12	12	177	181	15.4	15.7
4	18	20	671	786	37.3	39.3
9	32	33	1 585	1 671	49.5	50.6
16	57	58	2 883	2 946	50.2	50.7
25	86	89	4 549	4 807	52.8	54.0

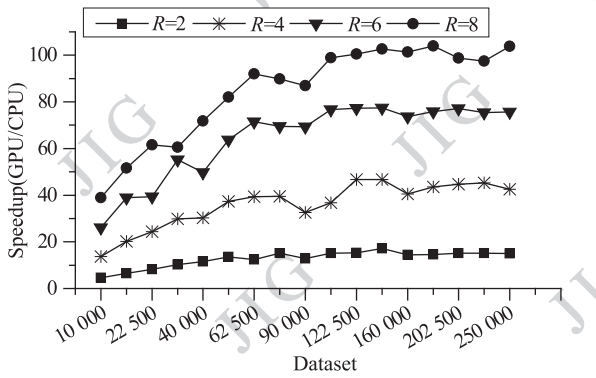


图 8　加速比增幅

Fig. 8　Speed up trends

当插值半径为 2 时,GPU 并行插值的效率能达到 CPU 算法效率的 3 ～18 倍;当插值半径分别为 4,6,8 时,加速比分别达到 10 ～55 倍,15 ～105 倍,40 ～180倍,也就是说,插值半径越大,GPU 并行的效率越高,因为插值半径增大意味着每个目标点需要对更大范围内的更多采样点进行数值运算。CPU 算法增加的迭代次数为所有目标点单独增加的迭代次数的总和,而对于 GPU 并行算法,整体增加的迭代次数仅为每个目标点增加的迭代计算。

对于相同插值半径,目标点越多,插值计算量越大,进而加速比也越高。但是,并行处理的性能提升倍数并非呈线性增长,随着输入计算量的增大,加速增幅趋于平缓。原因是当数据量不断增加时,主存与显存的通信时间将成为性能提升的瓶颈。

4　结　论

为了克服传统空间插值算法的低效问题,本文利用 GPU 通用计算架构,通过建立显存线性哈希索引及 CUDA 的多线程并行模式,实现了一种快速插值并行算法。实验表明该算法可视化效果较好,同时能达到几十倍甚至上百倍的加速比,插值性能提升显著,为并行计算的研究与实践提供一个创新性的尝试和良好的解决方案。对于一些实时性要求较高的应用场景,如 WebGIS 和图像处理等领域,该算法都具有一定的通用性和可行性。

参考文献(References)

[1] Moore P K. Interpolation error-based a posteriori error estimation for two-point boundary value problems and parabolic equations in one space dimension [J]. Numerische Mathematik, 2001, 90(1): 149-177.

[2] Blu T, Thevenaz P, Unser M. Linear interpolation revitalized [J]. Transactions on Image Processing, 2004, 13(5): 710-719.

[3] Cao Weiming. An interpolation error estimate in R^2 based on the anisotropic measures of higher order derivatives [J]. Mathematics of Computation, 2008, 77(261): 265-286.

[4] Geer D. For programmers, multicore chips mean multiple challenges [J]. Computer, 2007, 40(9):17-19.

[5] Komatitsch D, Michea D, Erlebacher G. Proting a high-order finite-element earthquake modeling application to NVIDIA graphics cards using CUDA [J]. Journal of Parallel and Distributed Computing, 2009, 69(5):451-460.

[6] NVIDIA company. CUDA homepage [EB/OL]. [2011 - 06 - 07]. http://nvidia.com/cuda, 2010.

[7] Agarwal P K, Krishnan S, Mustafa N H, et al. Streaming geometric optimization using graphics hardware [J]. Lecture Notes in Computer Science, 2003,2832: 544-555.

[8] Govindaraju N K, Lin M C, Manocha D. Fast and reliable collision detection using graphics processors [C]//Proceedings of the Twenty-First Annual Symposium on Computational Geometry. New York, USA: ACM, 2005: 384-385.

[9] Govindaraju N K, Raghuvanshi N, Manocha D. Fast and approximate stream mining of quantiles and frequencies using graphics processors [C]//Proceedings. of the 2005 ACM SIGMOD International Conference on Management of Data. New York, USA: ACM, 2005: 611-622.

[10] Li W, Fan Z, Wei X, et al. GPU-based flow simulation with complex boundaries [G]// Pharr M. Programming Techniques for High-Performance Graphics and General-Purpose Computation: GPU Gems 2. Addison-Wesley Professional, 2005: 747-764.

[11] Che S, Boyer M, Meng J Y, et al. A performance study of general-purpose applications on graphics processors using CUDA [J]. Journal of Parallel and Distributed Computing, 2008, 68(10): 1370-1380.

[12] Ortega L, Rueda A. Parallel drainage network computation on CUDA [J]. Computers & Geosciences, 2010, 36(2): 171-178.

[13] George Y L, David W W. An adaptive inverse-distance weighting spatial interpolation technique [J]. Computers & Geosciences, 2008, 34(9): 1044-1055.

© 中国图象图形学报版权所有