

基于 CUDA 的 IDW 并行算法及其实验分析

刘二永, 汪云甲

(中国矿业大学 环境与测绘学院, 徐州 221116)

摘要: 近些年来, 空间数据获取技术得到了迅猛的提高, 例如 LIDAR, 通常可以产生成千上万个点, 这对计算机的处理能力提出了挑战。最近, 图形处理器(GPU)的计算能力得到了巨大的提升, 致使 GPU 的通用计算引起了关注。GPU 是流处理器的集合, 最近的设备的流处理器超过 240 个, 浮点峰值比 CPU 快 10 多倍。在 GPU 上编程和编译的环境称计算统一设备架构(CUDA), 它提供了可以简单生成并行代码的途径。基于 CUDA 的并行计算, 在很多领域得到了应用, 但是在空间插值方面应用较少。反距离权插值(IDW)算法, 因容易计算, 在空间插值中经常被使用。然而, 当维数增加时, 提升计算时间是紧要的, 故本文提出 CUDA 的 IDW 并行算法。并在相同的条件下, 对比 CUDA 和 CPU 的算法的运行时间, 数值实验表明, CUDA 算法的运行速度是 CPU 算法的 6 倍左右。

关键词: IDW; 并行算法; CUDA; CPU

DOI: 10.3724/SP.J.1047.2011.00709

1 引言

反距离权(IDW)插值是空间插值中经常被使用的算法, 它运行速度快, 容易计算, 而且易给出直观的解释^[1]。然而, 随着问题维数的增加, 提升运算速度是至关重要的。并行计算是解决运算速度问题的重要途径^[2]。早期的 GPU (graphics processing unit) 是为图形计算而生的^[3], 最近的 GPU 已可进行数据计算, 称为 GPGPU (general-purpose computing on graphics processing units)。NVIDIA 公司于 2007 年正式发布的 CUDA (Compute Unified Device Architecture, 计算统一设备架构) 是第一种不需借助图形学 API 就可以使用类 C 语言进行通用计算的开发环境和软件体系, 它提供了一个编写运行在 GPU 上的并行代码的简捷环境。支持 CUDA 的 GPU 可以看成是一个由若干向量处理器组成的超级计算机, 性能也确实可以和小型的超级计算机相比。CUDA 已被应用于图像处理、生物计算、分子动力学计算等领域, 并在这些领域中对 CPU 获得了一到两个数量级的加速^[4]。近几年来, CUDA 也已经被用到了地学领域^[5-11]。肖汉和张祖勋提出了一种基于 GPGPU 的 CUDA 架构快速影像匹配并行算法, 速度比 CPU 的算法

提高了 7 倍^[3]。赵元等研究了并行蚁群算法及其在区位选址中的应用, 数值实验结果显示 GPU 并行算法对 CPU 串行算法加速比在 7.8 至 8.8 之间^[5]。但是, CUDA 的并行计算在空间插值上的应用仍然较少。本文提出 CUDA 的 IDW 并行算法, 并在相同的条件下, 通过数值实验与 CPU 算法作了对比分析。

2 IDW 与 CUDA 简析

2.1 IDW 算法

IDW 是一种常用而简便的空间插值方法, 它以插值点与样本点间的距离为权重进行加权平均, 离插值点附近的样本点赋予的权重越大, IDW 的插值公式如下^[6]

$$z_p = \frac{\sum_{i=1}^k z_i / d_i^\beta}{\sum_{i=1}^k 1 / d_i^\beta} \quad (1)$$

其中, z_p 是 p 点的插值的结果; z_i 是 p 点附近的第 i 个样本点 p_i ; k 是 p 点附近的用于插值的点的个数; d_i 是 p_i 到 p 的欧氏距离; β 通常取 2。本文中采用的取点方法是以待插点为原点在四个象限中各取一点, 共取四点进行插值。

收稿日期: 2011-04-19; **修回日期:** 2011-09-22.

基金项目: 国家自然科学基金项目(40971275, 51174287)。

作者简介: 刘二永(1978-), 男, 江苏徐州人, 博士生, 讲师, 研究方向: 空间分析与并行算法。E-mail: cumtley@hotmail.com

2.2 CUDA 模型

CUDA 是一种并行编程模型和软件环境。CUDA 编程模型将 CPU 作为主机(Host), GPU 作为协处理器(co-processor)或者设备(Device)。在一个系统中可以存在一个主机和若干个设备。如图 1 所示, 一个完整的 CUDA 程序是由一系列的设备端 kernel 函数并行步骤和主机端的串行处理步骤共同完成的。这些处理步骤会按照程序中相应语句的顺序依次执行, 满足顺序一致性。一个 kernel 函数中存在两个层次上的并行, 即 Grid 中的 block 间并行和 block 中的 thread 间并行。两层并行模型是 CUDA 最重要的创新之一^[4]。

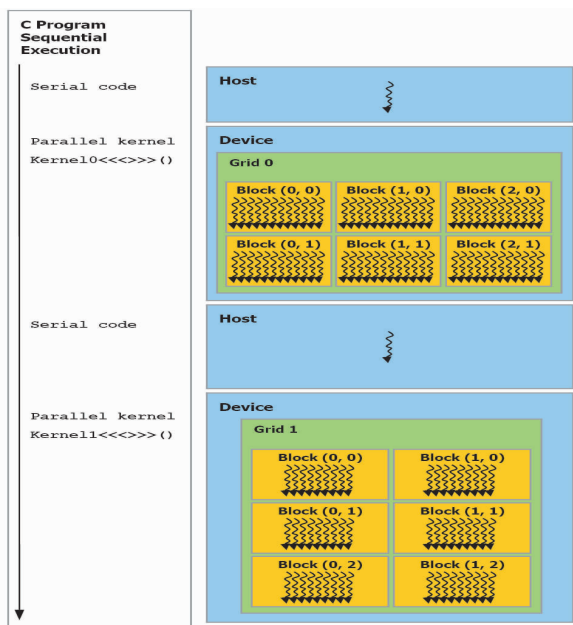


图 1 CUDA 编程模型^[7]

Fig. 1 CUDA programming model

3 基于 CUDA 的 IDW 算法

为了与后面提出的 CUDA 算法作对比, 首先给出 IDW 的 CPU 算法, 伪码见算法 1。

算法 1. void CPU_IDW()

1: define Integer WIDTH, HEIGHT //定义格网的维数

2: define float xllcorner, yllcorner, cellsize //定义格网的左下角的坐标和单元大小

3: define Integer SAMPLE_COUNT //定义样本个数

4: define struct XYZ{float X; float Y; float Z} //定义格网点的结构体

5: define struct XYZ samples_h[SAMPLE_COUNT] //定义格网点

6: define float results_h[WIDTH][HEIGHT] //定义存储结果的数组

7: input samples_h[SAMPLE_COUNT] //输入样本点数据

8: for i=0 to WIDTH-1 do

9: define float x←xllcorner+i*cellsize //计算待插值点的横坐标

10: for j=0 to HEIGHT-1 do

11: define float y←yllcorner+j*cellsize //计算待插值点的纵坐标

12: float results_h[i][j]=float CPU_interpolation_one(float x, float y,

const XYZ samples_h[SAMPLE_COUNT]) //插值格网点

13: end for

14: end for

15: output results_h[WIDTH][HEIGHT] //输出结果

算法 1 中的第 12 行的函数 CPU_interpolation_one 采用的是 2.1 节中的算法。下面给出 CUDA 的 IDW 算法的伪码, 见算法 2。

算法 2. void GPU_IDW()

1: define in GPU struct XYZ samples_d[SAMPLE_COUNT]

←CPU samples_h[SAMPLE_COUNT] //定义 GPU 格网点结构体, 并将 CPU 样本点写入 GPU。

2: define in GPU float results_d[WIDTH*HEIGHT] //定义 GPU 数组, 存储结果

3: define texture<float> samples_ptr_tex //定义纹理内存

4: cudaBindTexture(samples_ptr_tex, samples_d[SAMPLE_COUNT]) //绑定纹理内存

5: GPU_interplation_all(<<<>>>)(results_d[WIDTH*HEIGHT], samples_ptr_tex) //调用并行过程

6: cudaUnbindTexture(samples_ptr_tex) //释放纹理内存

7: results_d[xid]⇒results_h[WIDTH][HEIGHT] //将 GPU 结果写入 CPU。

算法 2 中相关的变量定义同算法 1, 其中, 第 5 行的 kernel 函数 GPU_interplation_all 的伪码见算法 3。

算法 3. void GPU_interpolation_all (results_d [WIDTH * HEIGHT], samples_ptr_tex)

```
1: define in GPU const int xid←blockDim. x
   * blockIdx. x+threadIdx. x//计算线程数
2: define in GPU const float x←xllcorner +
   (xid%WIDTH) * cellsize//计算待插值点的横坐标
3: define in GPU const float y←yllcorner +
   (xid/WIDTH) * cellsize//计算待插值点的纵坐标
4: results_d[xid]=float GPU_interpolation_
   one(float x, float y, samples_ptr_tex)//调用插值
   格网点函数
```

算法 3 的第 1 行中的 blockDim. x, blockIdx. x 和 threadIdx. x 都是 CUDA 中的内置变量,用于计算线程号。第 2 行和第 3 行中,%为取模运算,/为整除运算。第 4 行的函数 GPU_interpolation_one 采用的是 2.1 节中的算法,与算法 1 中的 CPU_interpolation_one 的区别在于取样本数据时是从纹理内存中取,在 Geforce 9800GT 显卡使用纹理后,速度提高了 10 多倍。

4 算法的数值实验与分析

本文以中国作为基本的实验区,采用 ASTER GDEM 为实验数据,从中选取了 3 个实验样区,大小分别是 1 000×1 000、2 000×2 000 和 3 000×3 000 网格,栅格分辨率约为 30m。为了试验的需要,用 ArcGIS 9.3 提取格网点,进入 Matlab 7.10 用双线性插值构造表面,并分别随机的选取30 000、120 000 和 270 000 个散点。最后用本文中的算法 1 和算法 2 插成 1 000×1 000、2 000×2 000 和 3 000

×3 000 格网(结果见图 2)。基于 CPU 和 GPU 的 IDW 算法的运行时间见表 1。实验环境为:联想商务机 M90, Intel (R) Core (TM) i5 cpu @ 2.67GHz 2.66GHz(4 核),4G 内存,NVIDIA GeForce 310 显卡。

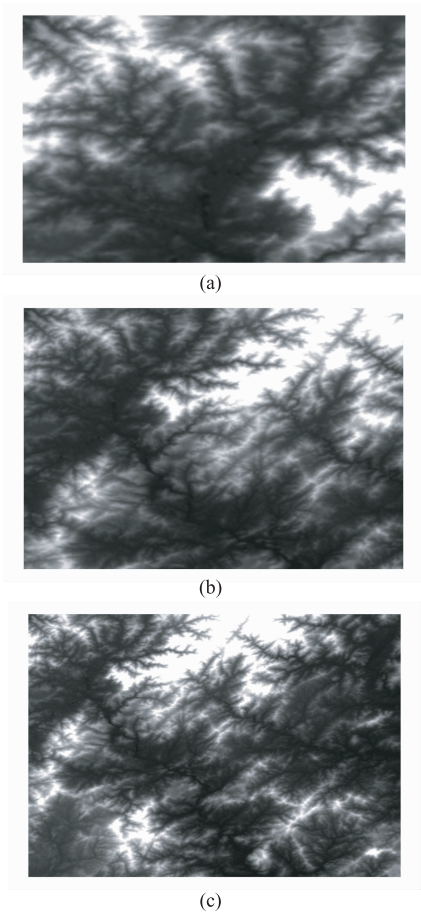


图 2 插成的 DEM
(a)样区 1;(b)样区 2;(c)样区 3
Fig. 2 Interpolation DEM of sample region 1(a), sample region 2(b), and sample region 3 (c)

表 1 三样区的运行时间

Tab. 1 Execution time of the three sample regions						
样区	网格大小	原始文件大小	采样点数量	CPU 运行时间	GPU 运行时间	加速比
样区 1	1 000×1 000	3.86M	30 000	2.9 分钟	27.7 秒	6.27
样区 2	2 000×2 000	15.7M	120 000	48.5 分钟	7.36 分钟	6.6
样区 3	3 000×3 000	34.9M	270 000	3.9 小时	42.2 分钟	5.5

从表 1 可看出,GPU 的 IDW 算法比 CPU 的速度快约 6 倍,起到了很好的加速效果。

5 结语

近几年来,随空间获取技术的提升,空间数据量变的越来越大,这也为计算机处理带来了一定的难度,而 CUDA 是解决这一问题的新途径。但是

CUDA 的并行计算在空间插值中的应用仍然较少。本文提出了基于 CUDA 的 IDW 的 GPU 算法,并在相同条件下,通过实验对比,结论是基于 GPU 的算法比 CPU 算法大约快 6 倍。

本文中的算法仍有改进的空间:(1)本文中采用的是在待插点的周围搜索四点,可以采用其他的更有效的搜索算法;(2)可以采用四叉树存储格网来节约存储量;(3)可以进一步改进成多个 GPU 或

者多机的并行算法。本文中的算法可以进一步推广到其他局部插值或者空间分析模型,这些都是进一步研究的课题。

参考文献:

- [1] Lu G Y, Wong D W. An Adaptive Inverse-distance Weighting Spatial Interpolation Technique[J]. *Computer & Geoscience*, 2008, 34: 1044 - 1055.
- [2] Ortega L, Rueda A. Parallel Drainage Network Computation on CUDA[J]. *Computer & Geoscience*, 2010, 36: 171 - 178.
- [3] 肖汉, 张祖勋. 基于 GPGPU 的并行影像匹配算法[J]. *测绘学报*, 2010, 39(1): 46 - 51.
- [4] 张舒, 褚艳利. GPU 高性能运算之 CUDA[M]. 北京: 中国水利水电出版社, 2009.
- [5] 赵元, 张新长, 康停军. 并行蚁群算法及其在区位选址中的应用[J]. *测绘学报*, 2010, 39(3): 322 - 327.
- [6] Guan Xuefeng, Wu Huayi. Leveraging the Power of

Multi-core Platforms for Large-scale Geospatial Data Processing: Exemplified by Generating DEM from Massive LIDAR Point Clouds[J]. *Computer & Geoscience*, 2010, 36: 1276 - 1282.

- [7] NVIDIA. NVIDIA CUDA 编程指南[CP/OL]. [2010-10-25]. http://www.nvidia.com/object/cuda_home_new.html.
- [8] 黄先锋, 程晓光, 张帆, 龚健雅. 基于边长比约束的离散点准确边界追踪算法[J]. *武汉大学学报. 信息科学版*, 2009, 34(6): 688 - 691.
- [9] 程林, 王美玲, 张毅. 一种基于 SuperMap GIS 的改进 Dijkstra 算法[J]. *地球信息科学学报*, 2010, 12(5): 649 - 654.
- [10] 陈冬平, 陈莹, 陈兴伟. 以 DEM 提取流域水系河源的最小误差分析[J]. *地球信息科学学报*, 2011, 13(2): 240 - 244.
- [11] 马建超, 林广发, 陈友飞, 陈俊明. DEM 栅格单元异质性对地形湿度指数提取的影响分析[J]. *地球信息科学学报*, 2011, 13(2): 157 - 163.

Parallel IDW Algorithm Based on CUDA and Experimental Analysis

LIU Eryong, WANG Yunjia

(School of Environment and Spatial Informatics, China University of Mining and Technology, Xuzhou 221116, China)

Abstract: In recent years, spatial data acquisition methods were significantly improved, such as LiDAR, which usually generated hundreds of millions of points. These amounts of datasets create great challenges to computation capacity of computers. In the past several years, the computing capacity of graphical processing unit (GPU) has improved significantly, too. General-purpose computing on GPUs has come into notice. GPUs are an aggregation of streaming processors. The amount of streaming processors in latest device exceeds 240. The peak floating-point operations per second of CPUs are ten times slower than that of GPUs now. A new software platform, called Computer Unified Device Architecture (CUDA), allows GPU programs to be developed in ANSI C. Parallel parts are worked on GPUs based on kernels, which are invoked by the CPU. Each kernel works on a grid of blocks, and each block is an array of threads. In the application process, each block is mapped to a multiprocessor, and each thread is mapped to a streaming processor. A typical CUDA program follows the flows as follow. First, the host function begins by locating one or more buffers in the GPU global memory and conveys the data to them. Then the CUDA program is started more times by appointing the number of threads per block. Finally, the results are transformed back to CPU memory. GPUs have been applied to solve many problems in signal processing, computational geometry and so forth. However, little has been used in spatial interpolation. CUDA Inverse-distance weighting (IDW) algorithm is the most frequently used model in spatial interpolation because it is relatively easy to compute. However, when dimensions increase, obtaining fast running time remains important. In this study, we explore the parallel algorithm for IDW, using the CUDA developed by NVIDIA. The main objective is to compare running times using CPUs versus GPUs under the same conditions. The numerical experiments show that processing speed of CUDA-based algorithm is 6 times faster than that of CPU-based method.

Key words: IDW; parallel algorithm; CUDA; CPU