DOI:10.13203/j.whugis20150361　　文章编号:1671-8860(2017)12-1688-08

一种面向 CPU/GPU 异构环境的协同并行空间插值算法

王鸿琰[1]　关雪峰[1,2]　吴华意[1,2]

1　武汉大学测绘遥感信息工程国家重点实验室,湖北 武汉,430079
2　地球空间信息技术协同创新中心,湖北 武汉,430079

摘　要:CPU/GPU 异构混合系统是一种新型高性能计算平台,但现有并行空间插值算法仅依赖 CPU 或 GPU 进行加速,迫切需要研究协同并行空间插值算法以充分利用异构计算资源,进一步提升插值效率。以薄板样条函数插值为例,提出一种 CPU/GPU 协同并行插值算法以加速海量激光雷达(light detector & ranger, LiDAR)点云生成数字高程模型(DEM)。通过插值任务的分解与抽象封装以屏蔽底层硬件执行模式的差异性,同时在多级协同并行框架基础上设计了 Greedy-SET 动态调度策略,策略顾及底层硬件能力的差异性,以实现异构并行资源的充分利用和良好负载均衡。实验表明,协同并行插值算法在高性能工作站上取得19.6倍的加速比,相比单一 CPU 或 GPU 并行算法,其效率提升分别达到 54%和 44%,实现了高效的协同并行处理。

关键词:CPU/GPU 异构环境;协同并行算法;空间插值;薄板样条插值函数;LiDAR 点云

中图法分类号:P208　　**文献标志码**:A

传感器技术的发展和数据采集能力的增强等导致空间数据规模爆炸式增长。以机载激光雷达(light detector & ranger, LiDAR)技术为例,一次飞行通常可获取数以千万计的高程数据点,已成为快速、精确获取地面三维数据的重要工具[1]。海量空间数据对传统 GIS 算法提出了严峻挑战,如传统空间插值算法面对海量点云数据已经难以满足快速分析的需求。

为此,已有大量研究利用多核 CPU 或众核 GPU 加速空间插值过程。多核 CPU 出现于 21 世纪初,已发展为成熟高性能计算平台,应用广泛。在多核 CPU 方面,文献[2]利用 OpenMP 对泛克里金插值算法进行加速,文献[3]利用 Intel 多线程函数库实现了反距离加权插值算法的快速求解。早期 GPU 是为图形处理而设计的,2007 年 NVIDIA 公司推出 CUDA(compute unified device architecture)通用编程模型,降低了 GPU 编程难度,推动了 GPU 通用计算的快速发展。目前 CUDA 已经广泛应用于应用数学[4]、物理学[5]、生物医药[6]、地震计算[7]等诸多领域。众核 GPU 具有计算能力强、带宽高、低能耗、性价比高等优点。鉴于 GPU 优秀的计算能力,更多的空间插值算法利用众核 GPU 进行加速,如反距离加权插值算法[8]、泛克里金插值算法[9]、普通克里金插值算法[10]、自然邻近插值算法[11]等,这些算法都基于 CUDA 进行并行改造,效率均有所提升。

当前计算平台基本同时配备了多核 CPU 和众核 GPU,CPU/GPU 异构混合系统凭借强劲计算能力、高性价比和低能耗等特点已经成为新型高性能计算平台。文献[8-11]中的空间插值算法仅采用单一并行开发模型实现,无法同时利用两种异构并行资源。因此,迫切需要研究协同并行算法以进一步提升插值效率。但是目前 CPU/GPU 异构混合系统缺乏完善统一的开发语言,现有编程模型不能完全屏蔽底层硬件的异构性,使得异构系统程序的开发难度大,开发效率低下[12]。底层硬件的异构性导致了 CPU、GPU 执行模式的差异性。同时,如何实现 CPU/GPU 高效协同是协同并行计算的另一关键问题[13],因而算法设计必须综合考虑算法流程和硬件能力,设计高效的任务调度策略以实现资源的充分利用和

收稿日期:2016-03-21
项目资助:国家自然科学基金(41301411);湖北省自然科学基金(2015CFB399)。
第一作者:王鸿琰,博士生,主要从事高性能地理计算研究。wanghongyan@whu.edu.cn
通讯作者:关雪峰,博士,讲师。guanxuefeng@whu.edu.cn

负载均衡。

本文以薄板样条函数插值算法为例,设计并实现了一种 CPU/GPU 协同并行插值算法以加速海量 LiDAR 点云生成数字高程模型。

1　CPU/GPU 异构系统及混合编程模型

在 CPU/GPU 异构系统中,CPU、GPU 具有不同的硬件架构和执行模式。CPU 由算术逻辑运算单元(arithmetic logic unit,ALU)、控制单元(control unit,CU)和高速缓存(cache)等构成。CPU 专为优化串行代码设计,其中核心晶体管主要用于复杂的控制单元和缓存以提高整体执行效率。与过去单核 CPU 相比,多核 CPU 实质是将多个处理核心进行封装或集成以实现更高的处理性能。传统串行程序必须经过并行改造才能利用多核 CPU 的计算资源。多线程并行计算主要通过两种方式实现,一是直接使用操作系统底层线程管理 API,如 Pthreads API;二是通过语言扩展或函数库等方法实现多线程并行计算,如 OpenMP 和 Intel 线程构建模块[14](threading building block, TBB)等。

对比 CPU,GPU 中晶体管主要用作 ALU 执行单元。以 NVIDIA Fermi 架构为例,一块 NVIDIA GPU 最多可包含 16 组流多处理器(streaming multiprocessor,SM),每组 SM 又包含 32 个流处理器(streaming processor,SP),数百个"核心"(SP)使得大规模并行计算成为可能。CUDA 是目前应用最为广泛的 GPU 编程模型。在 CUDA 模型中,CPU 作为主机(host),负责逻辑性强的事务处理和串行计算;GPU 则作为设备(device),专注于执行高度线程化的并行处理任务。运行在设备端的并行计算函数称为 Kernel,一个完整的 CUDA 程序包含主机端的串行处理步骤和一系列的设备端 Kernel 函数。Kernel 函数以线程网格(grid)的形式组织,线程网格由若干线程块(block)组成,而一个线程块又包含若干线程(thread)。Kernel 函数的执行单位是 block,逻辑上的 block、thread 与物理器件 SM 和 SP 相对应。每个 block 必须在一个 SM 上执行,而 block 内 thread 则被调度到相应的 SP 上执行。

虽然基于多核 CPU 和众核 GPU 的编程模型都已逐渐成熟,但针对异构混合系统仍然缺乏完善统一的编程模型。因此,目前开发 CPU/GPU 异构并行程序的主要方式仍然是混合编程模型,即把多核 CPU 和 GPU 上的成熟编程模型结合起来,以实现 CPU、GPU 协同并行计算,如典型的混合编程模型 OpenMP+CUDA[15]。

2　薄板样条函数插值算法

空间插值[16]法是指利用一组已知空间数据,通过建立一定函数关系来推算区域范围内其他任意未知点值的方法。薄板样条函数插值算法(thin plate spline,TPS)是常用的空间插值算法之一,它将插值问题模拟为一个金属薄板在空间点约束下的弯曲变形,通过最小化弯曲能量得到一个光滑曲面。在高程插值中,给定空间区域 R^2 内分布的 n 个已知点,其坐标表示为 (x_i,y_i,z_i),则通过最小化弯曲能量推导得 TPS 插值函数为:

$$z_i=a_0+a_1x_i+a_2y_i+\sum_{j=1}^{n}\lambda_j r_{ij}^2\log(r_{ij}) \tag{1}$$

式中,$\lambda_j(j=1,2,\cdots,n)$、$a_0$、$a_1$、$a_2$ 为待求解的插值参数;r_{ij} 表示两点间的欧氏距离;$\varphi(r_{ij})=r_{ij}^2\log(r_{ij})$ 表示薄板样条核函数。同时,TPS 算法必须满足边界条件(2),以保证在无限远处弹性变换为 0[17]:

$$\sum_{j=1}^{n}\lambda_j=0,\sum_{j=1}^{n}\lambda_j x_j=0,\sum_{j=1}^{n}\lambda_j y_j=0 \tag{2}$$

结合式(1)、(2),将 TPS 算法表示为矩阵形式:

$$\begin{pmatrix}0&\cdots&\varphi(r_{1n})&1&x_1&y_1\\ \varphi(r_{21})&\cdots&\varphi(r_{2n})&1&x_2&y_2\\ \vdots&\vdots&\vdots&\vdots&\vdots&\vdots\\ \varphi(r_{n1})&\cdots&0&1&x_n&y_n\\ 1&\cdots&1&0&0&0\\ x_1&\cdots&x_n&0&0&0\\ y_1&\cdots&y_n&0&0&0\end{pmatrix}\begin{pmatrix}\lambda_1\\ \lambda_2\\ \vdots\\ \lambda_n\\ a_0\\ a_1\\ a_2\end{pmatrix}=\begin{pmatrix}z_1\\ z_2\\ \vdots\\ z_n\\ 0\\ 0\\ 0\end{pmatrix} \tag{3}$$

因此,插值参数的求解转化为对线性方程组(3)的求解,将求得的插值参数和插值点坐标(x_i,y_i)代入式(1),即可求得区域内所有插值点的高程估计值。线性方程组求解和高程估值的计算复杂度分别为 $O(n^3)$ 和 $O(m\cdot n)$,其中 m 表示插值点个数。在实际应用中,随着样本点数 n 的增大,线性方程组(3)很快成为病态方程组,从而求得错误的插值参数,且求解线性方程组的时间随 n 增大而急剧增长,使其无法应对海量数据规模。为克服这一难题,本文采用局部邻域插值方式,即对每个插值点,算法搜索一定数量的邻近点进行参数解算并对该点估值,以此避免病态方程组的

产生并降低计算量。

3 TPS算法的协同并行化

3.1 TPS任务划分与抽象

TPS算法应用于LiDAR点云插值时，按执行流程可划分为空间划分、数据单元读取与任务封装、格网索引构建、TPS插值(包含邻近搜索)以及输出DEM 5个子任务。其中空间划分是协同并行的基础。本文采用局部邻域插值方式，弱化了空间区域关联，使得对原始数据进行空间划分成为可能；采用带重叠的规则格网划分，将海量空间数据划分为众多离散数据单元。空间划分粒度决定数据单元的大小和总量。相邻单元之间有数据重叠，重叠区域宽度设为邻近搜索半径，以确保边界点的临近搜索和插值后边界的连续性。

在插值过程中，直接按距离计算最邻近点非常耗时。为了节省计算资源和时间，本文对每个数据单元建立规则格网索引以加速邻近搜索过程。如图1所示，算法首先根据插值点P的坐标确定其所在网格，然后从该网格开始，从内层向外层进行邻近点搜索。在每层网格上，按照图1中顺序进行搜索。考虑到点云数据分布不均匀，本文采用固定搜索点数和固定搜索半径相结合的方法。根据经验统计值，一般取10～30个样本点参与插值较为合适[16]，而搜索半径的大小由点云分布的均匀程度和平均点距决定。

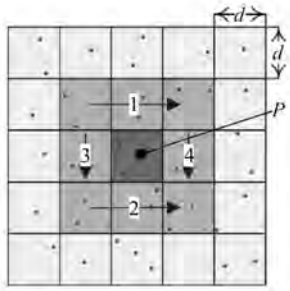

图1 格网索引与邻近搜索

Fig.1 Illustration of Grid Index and Neighbor Search

为屏蔽执行模式差异性，本文对计算任务进行抽象封装，设计了图2中的通用抽象任务类TPS-Task。其中，InBlock类包含数据单元的边界、点数以及点坐标等信息，而OutBlock类则表示输出数据单元，包含其边界、插值点高程等信息。算法执行时，数据单元读取到内存并封装为对应任务单元，任务单元负责为数据单元构建格网索引。根据处理器的空闲情况，任务调度器将插值任务调度到相应的CPU/GPU处理单元，任务单元与相应CPU/GPU插值函数实现了动态绑定。插值函数调用时通过配置CPU/GPU工作线程以适应不同性能的异构计算平台。

```
class TPS_Task
{
  public:
     InBlock*  ib;
     OutBlock* ob;
  public:
     void GridIndex(); //建立格网索引
     OutBlock* cpu_tps(InBlock* ib);
     OutBlock* gpu_tps(InBlock* ib);
     void Output(); //输出DEM
};
```

图2 TPS任务抽象

Fig.2 Abstraction of TPS Task

3.2 协同并行插值框架

如图3所示，空间划分形成的数据单元与插值计算封装组合成任务单元，算法通过构建一个由CPU、GPU共享可并发访问的先进先出(FIFO)队列对任务单元进行管理。队列长度可自由设定，算法通过限定队列长度以保持较低的内存占用。I/O线程不断读取数据，并将任务单元装入队列。任务调度器将任务单元调度到空闲的处理单元进行插值计算，数据读取与插值计算部分重叠，提高了整体执行效率。

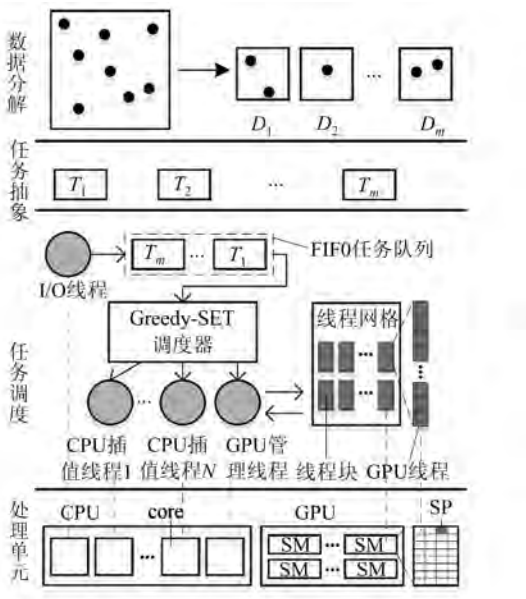

图3 协同并行插值框架

Fig.3 Framework of the Collaborative Parallel Interpolation

为充分挖掘异构平台的处理器性能，协同并行插值采用粗细粒度相结合的多级协同并行框架。在CPU端，调度器将不同任务单元调度到

多个 CPU 插值线程上进行并行处理,线程之间构成粗粒度并行。GPU 则只同时处理一个任务单元。在 GPU 端,单个插值点的插值计算作为原子任务映射到一个 GPU 线程上,而每个线程块负责一定数量插值点的插值计算。实际执行时,线程块作为执行单位映射到一个 SM 上执行,而块内线程又映射到 SM 相应的 SP 上执行,实现了精细粒度并行。由于算法要求对每个插值点进行邻近搜索,存储器访问频率很高。利用 GPU 线程切换零开销的特点,算法通过大规模线程并发来隐藏存储器访问延迟,从而使 GPU 保持较高的计算强度。

算法执行时生成多个 CPU 线程并行处理任务,线程资源可简单抽象为三元组[IO, G, C],各分量分别表示 I/O 线程数、GPU 管理线程数和 CPU 插值线程数。其中 1 个 I/O 线程负责数据读取和任务队列管理,1 个 CPU 线程作为 GPU 管理线程将数据传输给 GPU 并回收计算结果,N 个 CPU 线程作为 CPU 插值线程以利用多核 CPU 计算资源。CPU 插值线程和 GPU 管理线程可以单独或同时开启,N 值可以自由设定,这样算法可以形成不同执行模式,即串行模式([1, 0, 1])、单 CPU 并行模式([1, 0, $N>1$])、单 GPU 并行模式([1, 1, 0])和协同并行模式([1, 1, $N>1$])。

3.3 Greedy-SET 调度策略

由于点云分布的不均匀性,采用静态任务调度策略容易造成负载不均衡。针对任务单元计算量起伏不定的情况,一种简单的动态调度策略是用完即取的贪婪(Greedy)调度策略,即每次 CPU 插值线程或 GPU 完成相应任务单元的处理后,调度器立即将新任务调度到空闲的处理单元上,CPU、GPU 按各自计算能力处理一定数量的任务单元。然而,Greedy 策略是一种局部最优策略,在最后阶段容易造成负载不均衡的情况,尤其是当 CPU 和 GPU 性能悬殊时,往往会造成 GPU 闲置。且随着空间划分粒度增大,任务单元计算量增大,负载不均问题随之加重,从而降低整体执行效率。

为改善 Greedy 策略潜在的负载不均问题以使整体执行时间最短(shortest execution time, SET),本文基于 Greedy 策略设计了 Greedy-SET 策略。Greedy-SET 策略调度的依据是 GPU 和 CPU 插值线程对任务单元的平均处理时间。考虑到点云密度较高的特点,点云分布不均造成任务单元的计算量波动较小。如图 4 所示,Greedy-SET 策略根据处理器空闲情况将任务单元调度到 GPU 和 CPU,并更新平均插值时间。在最后阶段,即当 I/O 线程完成所有数据的读取后,若 CPU 线程平均插值时间小于剩余任务单元数与 GPU 平均插值时间的乘积,则 Greedy-SET 调度器将任务单元调度到相应的 CPU 线程进行处理,否则终止 CPU 插值线程。

```
while true do
  if task_queue is not empty then
    if GPU is available then
      dispatch task to GPU
      /*TPS interpolation on GPU*/
      update aver_gpu_time //GPU平均插值时间
      continue
    end
    if data reading is finished then
      if aver_cpu_time < queue_size*aver_gpu_time
        dispatch task to the CPU thread
        /*TPS interpolation on CPU*/
        update aver_cpu_time //CPU平均插值时间
        continue
      else
        kill the CPU thread
      end
    else
      dispatch task to the CPU thread
      /*TPS interpolation on CPU*/
      update aver_cpu_time
      continue
    end
  else
    break
  end
end
```

图 4　Greedy-SET 任务调度策略

Fig.4　Pseudocode of Greedy-SET Scheduling Strategy

4　实验及分析

4.1　实验配置与设计

硬件实验平台为一台高性能工作站,配置为:CPU 为 Intel Xeon E5-2620(2.00 GHz),双路 6 核,浮点运算性能为 192 Gflops,内存 96 GB。GPU 为 NVIDIA Tesla C2050 计算卡,包含 448 个 CUDA 核心,其核心频率为 1.15 GHz,双精度浮点性能峰值达 515 Gflops,带宽 144 GB/s,显存 3 GB GDDR5。实验开发环境与开发工具为操作系统 CentOS 6.5,Pthreads,Intel TBB 4.2 和 CUDA 5.0。

实验数据为美国 West Virginia Gilmer County 的 LiDAR 点云数据,采集时间为 2004-03-25～2004-04-07。数据面积为 336 km^2,点数为 8.8 亿,点距为 1.4 m,大小为16.4 GB。根据经验公式[3,16],协同并行算法预先设定邻近搜索的半径为 20 m,搜索不超过 20 个样本点参与插值。算法设置输出 DEM 分辨率为2 m。

本文主要设计了4组对比实验，即串行模式(S-TPS)、CPU并行模式(C-TPS)、GPU并行模式(G-TPS)和协同并行模式(CG-TPS)的性能对比实验。对于S-TPS，CPU插值线程数 $N=1$，GPU管理线程关闭；对于C-TPS，CPU插值线程数 N 等于当前平台的CPU核心数减2，GPU管理线程关闭；对于G-TPS，CPU插值线程数 N 为0，GPU管理线程开启，即所有插值任务都由GPU完成。CG-TPS中CPU插值线程数 N 等于当前平台的CPU核心数减2，GPU管理线程开启。对比实验旨在验证CG-TPS相比于S-TPS模式以及单一并行模式的效率提升。在每组实验中，设定不同的空间划分粒度，记录4种模式在不同划分粒度下的插值时间以验证数据分块尺寸对4种模式的效率影响。另外，本文在CG-TPS模式下对比了Greedy策略和Greedy-SET策略的性能表现，旨在验证Greedy-SET调度策略对负载不均问题的改善作用。本文仅采用加速比作为并行算法的性能衡量指标，而CG-TPS相对于C-TPS和G-TPS的效率提升则以百分比形式给出，分别记为 $P_{\text{CG-C}}$、$P_{\text{CG-G}}$：

$$\begin{cases} P_{\text{CG-C}} = \dfrac{T_{\text{C-TPS}} - T_{\text{CG-TPS}}}{T_{\text{C-TPS}}} \\ P_{\text{CG-G}} = \dfrac{T_{\text{G-TPS}} - T_{\text{CG-TPS}}}{T_{\text{G-TPS}}} \end{cases} \tag{4}$$

式中，分母分别表示C-TPS和G-TPS的插值时间，分子则表示CG-TPS相比两种并行模式节省的时间。由于CPU/GPU协同并行存在调度开销和负载问题，$T_{\text{CG-TPS}}$ 值大于将C-TPS和G-TPS进行简单联合的理论值，即 $P_{\text{CG-C}}$ 与 $P_{\text{CG-G}}$ 之和小于1，且其值越趋近于1表明负载越均衡。

表1 高性能工作站上各执行模式在不同空间划分粒度下的插值时间

Tab.1 Execution Time of Different Parallel Modes with Different Block Granularities on the Workstation

空间划分粒度/m	S-TPS/s	C-TPS/s	G-TPS/s	CG-TPS (Greedy)/s	CG-TPS (Greedy-SET)/s
500	15 003.5	1 629.8	5 327.6	1 293.8	1 285.4
1 000	15 000.8	1 632.9	2 380.6	984.2	981.5
1 500	14 970.7	1 635.4	1 762.9	869.2	862.6
2 000	14 999.0	1 643.5	1 522.1	822.1	817.9
2 500	14 991.0	1 648.5	1 448.6	802.1	787.8
3 000	15 044.7	1 675.1	1 371.1	790.1	767.0
3 500	15 034.5	1 674.6	1 324.3	787.8	773.2
4 000	15 101.0	1 689.3	1 310.1	827.3	774.3

4.2 实验结果与分析

表1列出了高性能工作站上算法各执行模式在不同空间划分粒度下的插值时间结果。图5则反映了不同并行模式取得的加速比。

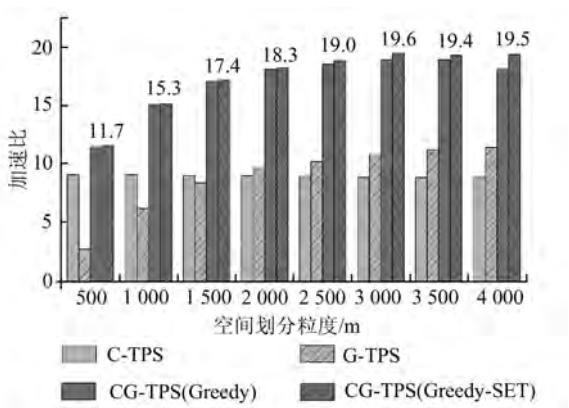

图5 不同并行模式在高性能工作站获取的加速比

Fig.5 Speedup of Different Parallel Modes on the Workstation

在高性能工作站上，S-TPS耗时约250 min，在一定范围内不受空间划分粒度影响，当粒度大于3 000 m后，其插值时间略有增长。C-TPS平均取得约9倍的加速比，插值时间随空间划分粒度增大表现出缓慢增长的趋势。G-TPS受空间划分粒度影响很大，如单元尺寸为500 m和4 000 m时，算法插值时间相差约4倍。G-TPS呈现出随空间划分粒度越大效率越高的变化趋势，其加速比首先快速增长随后趋于平稳，并最终取得11.5倍的加速比。此时，G-TPS相比C-TPS的效率提升较小，与对应GPU和CPU的浮点运算性能相差较大，说明频繁的显存访问很大程度上限制了GPU计算性能的发挥，这是由算法本身特点和GPU存储墙问题共同导致的。CG-TPS性能同样受空间划分粒度影响，且在一定粒度范围内与G-TPS表现出相似的加速比变化趋势，随后其插值时间反而增长，导致加速比降低。在不同划分粒度下，基于Greedy-SET策略的CG-TPS插值时间均优于基于Greedy策略的CG-TPS。当空间划分粒度取3 000 m时，CG-TPS(Greedy-SET)实现了19.6倍的最高加速比，此时协同并行算法在12.8 min内完成8.8亿个点的插值计算，并最终生成一幅大小为27 500×30 500的栅

格 DEM 图像,如图 6 所示。

图 6　协同并行 TPS 算法插值生成的栅格 DEM

Fig.6　Raster DEM Interpolated by the Collaborative Parallel TPS Algorithm

为最大化 CG-TPS 的总体执行效率,协同并行算法必须保证计算资源的充分利用和良好的负载均衡。在资源利用方面,对于 CPU 插值线程,适当数量的线程配置即可保证 CPU 资源的高效利用。而 GPU 插值效率受空间划分粒度影响较大,主要原因是由数据往返传输产生的通信开销和由数据量变化引起的 GPU 资源浪费。图 7 反映了一次 GPU 插值过程中数据传输和 TPS 插值计算的平均耗时情况。当数据单元尺寸为 500 m 时,TPS 插值耗时 0.13 s,数据传输耗时0.47 s,插值计算的时间占比仅为 21%。此时,细粒度空间划分产生了数量更多的数据单元,频繁的数据传输导致了大量的通信开销。同时,细粒度空间划分产生的相应数据单元的计算量较小,GPU 计算资源难以充分利用,最终导致 GPU 有效利用率不高。随着空间划分粒度增大,数据单元数量显著减少,而单次数据传输的时间增幅却很小,通信总开销降低。且更大尺寸的数据单元使得更多的时间和计算资源集中在插值计算上,有效提高了 GPU 的资源利用率。然而,单方面增大空间划分粒度并不能保证 CG-TPS 总体执行效率的提升。C-TPS 的插值时间变化说明,空间划分粒度越大,负载不均衡问题越严重。Greedy 策略虽然不用考虑数据单元本身的计算量波动问题,但在最后阶段往往会闲置 GPU,导致负载不均问题。而 Greedy-SET 策略一定程度上改善了这种负载不均问题,尤其在空间划分粒度较大时,其改善效果更加明显。因此,适当大的空间划分粒度是最大化 CG-TPS 执行效率的关键。

合理的空间划分粒度和任务调度策略能实现异构计算资源的充分利用和高效协同。图 8 反映了高性能工作站上 CG-TPS(Greedy-SET)相比 C-TPS 和 G-TPS 的效率提升情况。随着空间划分粒度的增大,并行资源尤其是 GPU 计算资源逐渐得到充分利用,使得 $P_{CG\text{-}G}$ 值减小,$P_{CG\text{-}C}$ 值增大。当空间划分粒度为 3 000 m 时,CG-TPS(Greedy-SET)取得的总体执行效率趋于最优。此时,相比于 C-TPS 和 G-TPS,CG-TPS(Greedy-SET)分别取得 54%和 44%的效率提升,实现了异构并行资源的充分利用和良好的负载均衡。

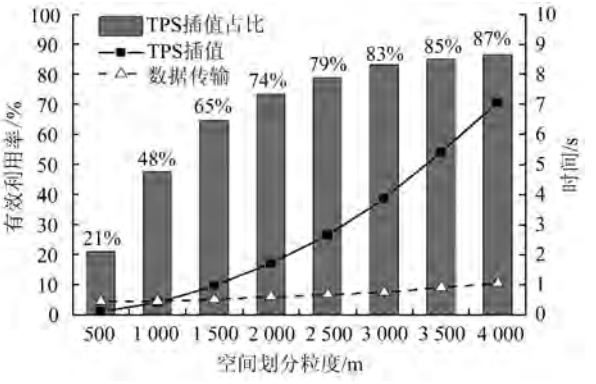

图 7　GPU 有效利用率

Fig.7　Utilization Ratio of GPU Resources

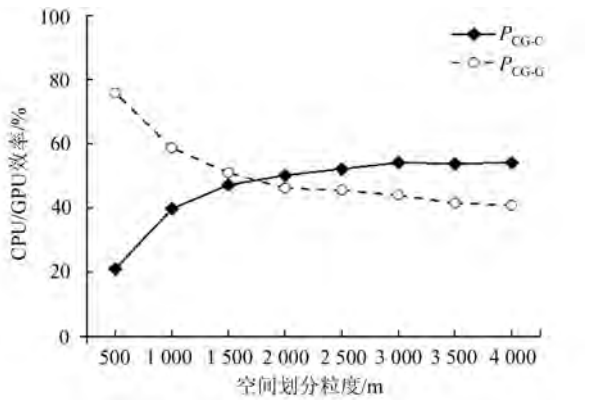

图 8　CG-TPS(Greedy-SET)相比 C-TPS 和 G-TPS 的效率提升

Fig.8　Efficiency Improvement of CG-TPS (Greedy-SET) Compared with C-TPS and G-TPS

5　结　语

针对海量空间数据插值效率低下和混合异构环境资源浪费的问题,本文以 TPS 算法为例设计并实现了一种 CPU/GPU 协同并行插值算法。该算法对插值任务进行抽象封装,屏蔽了底层硬件的执行模式差异性。结合多级协同并行框架,本文设计的 Greedy-SET 调度策略实现了计算资源的充分利用和良好负载均衡。相比基于多核 CPU 或众核 GPU 的单一并行插值算法,CPU/GPU 协同并行插值算法取得了显著的性能提升,证明了该算法的高效性。

本文设计的协同并行插值算法仍有继续深入研究和改进之处,尤其是适当的空间划分粒度无法预先测定,从而限制了算法的应用。建立精确的性能模型,并通过训练可以确定合适的空间划分粒度,而研究基于处理器能力感知的、自适应的数据划分方法有利于实现CPU、GPU更合理的利用和更好的负载均衡。

参 考 文 献

[1] Brovelli M, Cannata M. Digital Terrain Model Reconstruction in Urban Areas from Airborne Laser Scanning Data: the Method and an Example for Pavia (Northern Italy) [J]. *Computers & Geosciences*, 2004, 30(4): 325-331

[2] Cheng T, Li D, Wang Q. On Parallelizing Universal Kriging Interpolation Based on OpenMP[C]. International Symposium on Distributed Computing and Applications to Business Engineering and Science (DCABES), Hong Kong, China, 2010

[3] Guan X, Wu H. Leveraging the Power of Multi-core Platforms for Large-Scale Geospatial Data Processing: Exemplified by Generating DEM from Massive LiDAR Point Clouds [J]. *Computers & Geosciences*, 2010, 36(10): 1 276-1 282

[4] Sharma G, Agarwala A, Bhattacharya B. A Fast Parallel Gauss Jordan Algorithm for Matrix Inversion Using CUDA [J]. *Computers & Structures*, 2013, 128: 31-37

[5] Preis T, Virnau P, Paul W, et al. GPU Accelerated Monte Carlo Simulation of the 2D and 3D Ising Model [J]. *Journal of Computational Physics*, 2009, 228(12): 4 468-4 477

[6] Jeon Y, Jung E, Min H, et al. GPU-Based Acceleration of an RNA Tertiary Structure Prediction Algorithm [J]. *Computers in Biology and Medicine*, 2013, 43(8): 1 011-1 022

[7] Li B, Liu G F, Liu H. A Method of Using GPU to Accelerate Seismic Pre-Stack Time Migration [J]. *Chinese Journal of Geophysics*, 2009, 52(1): 242-249

[8] Liu Eryong, Wang Yunjia. Parallel IDW Algorithm Based on CUDA and Experimental Analysis [J]. *Journal of Geo-information Science*, 2011, 13(5): 707-710(刘二永, 汪云甲. 基于CUDA的IDW并行算法及其实验分析[J]. 地球信息科学学报, 2011, 13(5): 707-710)

[9] Cheng T. Accelerating Universal Kriging Interpolation Algorithm Using CUDA-Enabled GPU [J]. *Computers & Geosciences*, 2013, 54(2 013): 178-183

[10] De Ravé E G, Jiménez-Hornero F J, Ariza-Villaverde A B, et al. Using General-Purpose Computing on Graphics Processing Units (GPGPU) to Accelerate the Ordinary Kriging Algorithm [J]. *Computers & Geosciences*, 2014, 64(2 014): 1-6

[11] Beutel A, M∅lhave T, Agarwal P K. Natural Neighbor Interpolation Based Grid DEM Construction Using a GPU[C]. The 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems, San Jose, USA, 2010

[12] Ju Tao, Zhu Zhengdong, Dong Xiaoshe. The Feature, Programming Model and Performance Optimization Strategy of Heterogeneous Many-Core System: A Review [J]. *Acta Electronica Sinica*, 2015, 43(1): 111-119(巨涛, 朱正东, 董小社. 异构众核系统及其编程模型与性能优化技术研究综述[J]. 电子学报, 2015, 43(1): 111-119)

[13] Lu Fengshun, Song Junqiang, Yin Fukang, et al. Survey of CPU/GPU Synergetic Parallel Computing [J]. *Computer Science*, 2011, 38(3): 5-9(卢风顺, 宋君强, 银福康, 等. CPU/GPU协同并行计算研究综述[J]. 计算机科学, 2011, 38(3): 5-9)

[14] Reinders J. Intel Threading Building Blocks: Outfitting C for Multi-core Processor Parallelism[M]. Sebastopol: O'Reilly Media Inc., 2007

[15] Jang H, Park A, Jung K. Neural Network Implementation Using CUDA and OpenMP[C]. International Conference on Digital Image Computing: Techniques and Applications (DICTA), Canberra, Australia, 2008

[16] Mitas L, Mitasova H. Spatial Interpolation [J]. *Geographical Information Systems: Principles, Techniques, Management and Applications*, 1999, 1: 481-492

[17] Terzopoulos D. Regularization of Inverse Visual Problems Involving Discontinuities [J]. *Pattern Analysis and Machine Intelligence*, 1986, PAMI-8(4): 413-424

A Collaborative Parallel Spatial Interpolation Algorithmon Oriented Towards the Heterogeneous CPU/GPU System

WANG Hongyan[1]　*GUAN Xuefeng*[1,2]　*WU Huayi*[1,2]

1　State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University, Wuhan 430079, China

2　Collaborative Innovation Center of Geospatial Technology, Wuhan 430079, China

Abstract: Nowadays the heterogeneous CPU/GPU systems become ubiquitous, but most of current parallel spatial interpolation algorithms exploit only one type of computation units to speedup the calculation and thus it results in parallel resources wasted. To address this problem, a collaborative parallel thin plate spline interpolation algorithm is proposed in this paper to accelerate DEM generation from massive LiDAR point clouds. In this collaborative parallel algorithm, the input point clouds are firstly decomposed into a collection of discrete blocks and encapsulated as general task objects to shield the heterogeneous execution models of different processing units. And then a special scheduling algorithm, named Greedy-SET, is also proposed to achieve better load balance based on the computing capabilities of CPU and GPU. Experimental results demonstrate that the proposed collaborative parallel algorithm can achieve the highest speedup times of approximately 19.6. The performance improvement ratios compared with pure CPU and GPU parallel algorithms are 54% and 44% respectively.

Key words: CPU/GPU; collaborative parallel algorithm; spatial interpolation; thin plate spline; LiDAR point clouds

First author: WANG Hongyan, PhD candidate, specializes in high performance geocomputation. E-mail: wanghongyan@whu.edu.cn

Corresponding author: GUAN Xuefeng, PhD, lecturer. E-mail: guanxuefeng@whu.edu.cn

Foundation support: The National Natural Science Foundation of China, No. 41301411; Natural Science Foundation of Hubei Province, No. 2015CFB399.