基于 CUDA 的地震数据层位面自动追踪算法

魏 强,唐 杰
(南京大学 计算机科学与技术系,江苏 南京 210093)

摘 要:层位面追踪是三维地震数据解释的一项关键操作,对于地质勘探和油藏工程有重要的指导意义。传统的层位面自动追踪算法采用的是种子点蔓延法,该方法是一个不断进行迭代计算的过程。由于三维地震数据的数据量很大,传统的基于 CPU 的层位面自动追踪算法的计算效率不高。文中提出了一种基于 CUDA 的三维层位面自动追踪算法。该算法对种子点蔓延法进行了改进,实现了层位面自动追踪的 GPU 并行解决方案。实验表明,与传统的基于 CPU 的层位面自动追踪算法相比,本算法的计算效率有明显的提升。

关键词:CUDA;层位面; 地震数据处理; 自动追踪算法
中图分类号:TP301.6 **文献标识码**:A **文章编号**:1673-629X(2012)09-0001-04

CUDA-based Automatic Seismic Horizon Tracking Algorithm

WEI Qiang,TANG Jie
(Department of Computer Science and Technology, Nanjing University,Nanjing 210093,China)

Abstract:Horizon tracking is an important operation in 3-D seismic data processing, and the tracked horizon can be used by the experts to detect the earth's subsurface structure. Processing speed is a major bottleneck in seismic horizon tracking since the data size of 3-D seismic data is usually very huge. In this paper,present a new automatic horizon tracking algorithm, which is based on CUDA technique. The algorithm fully utilizes the many-core structure and the high performance computing ability of GPU, realizing a large scale parallel method for horizon tracking. The results of experiments show that the algorithm has an obvious speedup than the traditional CPU-based horizon tracking algorithm.

Key words:CUDA;horizon; seismic data processing;automatic tracking algorithm

1 概 述

地震解释工作是地震勘探中的一个重要环节。将野外获得的原始地震资料进行数据处理之后,得到可供解释的地震资料。解释人员根据地震波传播规律,对这些地震资料进行分析、研究,从而达到了解、推断地下构造和岩层性质的目的[1]。

层位面是地下不同岩层的分界面,对地质勘探、油藏工程有重要的指导意义。从三维地质数据中自动识别追踪层位面,可以减少人为干预,缩短地质解释时间。Faraklioti 等[2]提出了使用图像处理技术进行三维层位面追踪的算法,李雪峰等[3]提出了利用相干体[4]的剖面特征进行层位面追踪解释的方法,这些算法在实际应用中取得了良好的效果。但是随着地震勘探和数据采集技术的进步,三维地震数据的精度越来越高,数据量也越来越大,一块地震数据往往达到几十甚至上百 GB。同时,地震数据处理算法的计算量大,如何提高地震数据的处理速度是现在三维地震数据处理的研究热点之一。使用 CPU 集群来处理地震数据是一种解决方案,但是需要投入较高的成本,且降低了设备的可移动性。

CUDA(Compute Unified Device Architecture)[5]是 nVidia 推出的一种基于 GPU 的通用并行计算架构,该架构通过利用 GPU 强大的并行计算能力,可大幅提升计算性能。CUDA 的开发语言以标准的 C 语言为基础,并添加了一些简单的扩展,开发人员无需像开发传统的 GPGPU 程序那样将计算任务转换为图形渲染任务,从而大大降低了利用 GPU 进行并行程序开发的门槛。因此,CUDA 自推出之后就受到了软件开发人员和科研人员的热捧,并在图像处理、视屏解码、光线追踪、物理学模拟、计算生物学和化学等领域得到了广泛的应用[6]。三维地震数据处理计算具有较高的并行性且计算密度大,很适合利用 GPU 来进行并行运算,因此 CUDA 在地震数据处理方面也得到了广泛的应用。Bernard 等[7]利用 CUDA 技术实现了基于 GPU 集群

收稿日期:2012-01-15;**修回日期**:2012-04-24
基金项目:国家科技重大专项(2011ZX05035-004-004HZ)
作者简介:魏 强(1988-), 男,硕士研究生,研究方向为计算机图形学;唐 杰,博士,副教授,CCF 会员,研究方向为计算机图形学、三维建模。

的地下地质构造成像算法,与基于 CPU 集群的传统算法相比,计算速度取得了大约一个数量级的提升,同时节省了大量的运营成本。

文中提出了一种基于 CUDA 的层位面自动追踪算法,解决了蔓延法并行性不明显的不足,利用 GPU 强大的并行计算和浮点计算能力对传统算法进行提速。实验表明:文中算法与传统的 CPU 版本算法相比,计算效率有明显的提升。

2 基于 CUDA 的层位面自动追踪算法

同向轴自动拾取[8,9]是一种有效的层位面追踪算法。同相轴是地震记录上波的相同相位的连接线。当地下岩层存在分界面(层位面)时,返回地面的反射波就会引起地面振动,一个排列上所有的地震道都会跳动,这时就会在地震记录上出现波峰或波谷构成的同相轴[1]。因此同向轴可以作为层位追踪的依据。

同向轴自动拾取层位面追踪算法的基本思想如下:首先人工选择若干种子点,以确定需要追踪的层位面。然后从种子点所在的道向相邻道进行“蔓延”遍历,遍历过程中需要根据一定的约束条件在相邻道上选择合适的数据点,该数据点将作为追踪的结果加入层位面,并将该数据点作为新的种子点,继续进行遍历直到所有的道都被遍历到。

这里使用的基本约束条件包括:

(1) 波形的相似性:同一层位面在相邻道上的数据点的波形的相位应该相同,振幅应该相近。

(2) 空间位置的约束条件:同一层位面的相邻道上的数据点在空间位置上应该足够近,即相邻道上的数据点的深度值应该相近。

同向轴自动拾取算法可以划分为两个基本操作:第一个操作是扩展操作,其主要工作是从已经遍历过的道向其周围尚未遍历的道进行“蔓延”遍历。第二个阶段是调整操作,其主要工作是在新“蔓延”到的道上选择满足约束条件的数据点。算法终止的条件是遍历完所有的道,在遍历完所有的道之前,需要迭代地执行上述两个阶段的操作。

同向轴自动拾取算法虽然能有效地追踪出层位面,但是由于该算法是一个不断进行迭代的过程,导致传统的基于 CPU 的串行算法的运行效率不高,同时,由于蔓延算法的特点,其并行性并不明显,需要在算法上仔细考虑,才能充分发挥 GPU 的高度并行计算能力。此外,对地质曲面的解释工作是一个尝试并逐步优化的过程,地质解释人员需要对追踪出的层位面进行局部调整,这时需要改变自动追踪算法的参数并重新执行追踪算法,以优化追踪结果。如果追踪算法的运行时间较长,将严重影响地震解释工作的效率。文中对同向轴自动拾取算法进行了改进,实现了基于 CUDA 的层位面自动追踪算法,显著地提高了同相轴自动拾取算法的运行效率。

2.1 算法概述

同向轴拾取算法是一个不断进行迭代的过程,将所有的工作都交给 GPU 来完成是比较困难的。该算法中最为费时的部分是扩展操作和调整操作,两个操作都具有一定的并行性,所以文中算法利用 CUDA 对这两个操作进行了实现,以利用 GPU 的并行计算能力对算法进行提速。每一次迭代都会利用 GPU 来执行这两个操作,而算法的迭代终止条件的判断交给 CPU 来完成。

在每一次迭代中,并非所有的道都需要进行扩展操作和调整操作。这两个操作的计算较复杂,如果为每一道都分配一个 CUDA 线程来执行扩展操作和调整操作,将造成计算资源的严重浪费。为了解决这个问题,文中算法在进行扩展操作和调整操作之前,进行了“压缩”操作。“压缩”操作从所有道中筛选出当前迭代中需要进行扩展操作和调整操作的道,只需为“压缩”操作筛选出的道分配 CUDA 线程,这样可以减少不必要的运算。

文中算法的伪代码如算法 1 所示。其中,Seeds 是手工选择的种子点。Variance 是同一层位面上相邻道数据点振幅的最大允许偏差值,用来确保相邻道的数据点的振幅足够相近。这里选择了多个不同的偏差值标准:最初选择较小的偏差值进行追踪,如果所有的道都被追踪到了,则算法结束,否则保留追踪到的数据点作为种子点,并选择较大一些的偏差值继续进行追踪。偏差值可以选择振幅的绝对偏差值或相对偏差值,本算法中选择的是相对偏差值。Frontier 记录了当前遍历到的道的标号,NewFrontier 记录了从当前道新遍历到的道的标号。Tracked 是每个道的遍历状态标志,如果该道被遍历过,则其标志为 1,否则标志为 0。CUDAHorizonExtender 是执行扩展操作的 CUDA kernel function。CUDAHorizonAdjuster 是执行调整操作的 CUDA kernel function。CUDACompactFrontier 是执行“压缩”操作的 CUDA kernel function, CUDACompactFrontier 的第一个参数是“压缩”的输出结果,第二个参数是“压缩”操作的输入。输入是一个 unsigned int 类型的数组 In,其值为 0 或 1。输出是一个 int 类型的数组 Out, Out[i] 记录的是 In 数组中第 i 个 1 的下标。“压缩”操作可以利用 CUDA 的 scan primitive[10] 操作实现。

2.2 扩展操作

扩展操作的工作是从已经遍历的道向其邻域内未遍历的道进行遍历,邻域可以选择 4-邻域或 8-邻域。

其遍历流程类似于图的广度优先遍历(BFS)。可以构造一个与之对应的图:为每一个道构造一个顶点 i,为任意两个相邻的道 i 和 j 构造一条边 (i,j)。对于给定的顶点 i,可以根据邻域的定义直接确定与 i 相关的边,因此无需显式地记录图的边。从种子点所在道对应的顶点 s 开始进行广度优先遍历,直到遍历完所有的顶点,由于顶点和道是一一对应的,所以这样也遍历完了所有的道。

算法 1 GPUHorizonTracker(Seeds)

```
Frontier ← Seeds
for all v ∈ Variance do
  while Frontier ≠ ∅ do
    for all f ∈ Frontier in parallel do
      Invoke CUDAHorizonExtender(f, v)
    end for
    Invoke CUDACompactFrontier(NewFrontier, BeNewFrontier)
    for all f ∈ NewFrontier in parallel do
      Invoke CUDAHorizonAdjuster(f)
    end for
    Invoke CUDACompactFrontier(Frontier, BeFrontier)
  end while
  Frontier ← Tracked
end for
```

P. Harish[11] 等实现了基于 CUDA 的广度优先遍历算法。我们在其基础上实现了扩展操作的算法 CUDAHorizonExtender,其伪代码如算法 2 所示。这里没有使用队列来实现 BFS 算法,而是使用了三个标志数组 BeFrontier 、BeNewFrontier 和 Tracked。由于 BFS 采用层次遍历的方式对图进行遍历,所以这里使用数组 BeFrontier 来记录当前遍历层次的顶点,使用另外一个标志数组 BeNewFrontier 来记录新遍历到的顶点,新遍历到的顶点将作为下一层次的顶点。对于一个顶点 f,如果 f 是当前层次中的顶点,其标志 BeFrontier 为 1,否则为 0。Tracked 记录的是所有顶点的遍历状态,如果顶点 n 已经被遍历过了,则 Tracked 为 1,否则为 Tracked 为 0。当从当前顶点 f 遍历到一个新的顶点 n 时,需要查看其 Tracked 标志是否为 0,若 Tracked$[n]$ 为 0 ,则将 n 加入下一遍历层次的顶点数组,并设置 BeNewFrontier$[n]$ 为 1。

对于新遍历到的顶点 n, 这里使用了两个数组 Depth 和 Threshold 来记录新遍历到的顶点对应道的约束条件。为了方便描述,下面将当前层次的道称为源道,从源道新遍历到的道称为目标道。Depth 取对应源道上种子点的深度值,目标道上的数据点需要在 Depth 附近的一个深度范围内进行搜索,以满足深度约束条件。Threshold 记录了目标道上数据点需要满足的振幅约束条件,即振幅的阈值。如果源道上种子点的振幅值为 a,且选择的相对偏差值为 v,那么振幅阈值为 $a \times (1 - v)$。 这两个值将作为调整阶段选择目标道数据点的约束条件。

算法 2 CUDAHorizonExtender(f, v)

```
targetDepth ← ResultPos[f]
targetThreshold ← ResultVal[f] × (1 − v)
for all n ∈ neighborhood of f do
  if Tracked[n] = 0 then
    BeNewFrontier[n] ← 1
    Depth[n] ← targetDepth
    Threshold[n] ← targetThreshold
  end if
end for
```

2.3　调整操作

调整操作的工作是在目标道上找到满足约束条件的数据点,即需要追踪的层位面上的点。

为了减少执行调整操作的 CUDA 线程数量,避免大量线程执行不必要的操作,在进行调整操作之前先对 BeNewFrontier 进行了“压缩”操作:对于道 i,如果其标志 BeNewFrontier$[i]$ 为 1,则将 i 加入数组 NewFrontier 。NewFrontier 数组记录了所有的目标道。对 BeNewFrontier 进行“压缩”操作之后,只需要对 NewFrontier 中记录的目标道创建 CUDA 线程来执行 CUDAHorizonAdjuster 操作。由于调整操作的计算复杂,所以执行“压缩”操作所带来的开销是值得的。

调整操作的伪代码如算法 3 所示。如果在目标道上找到了满足约束条件的数据点,则设置该道的 BeFrontier 和 Tracked 标志为 1。在调整操作结束之后,需要对 BeFrontier 进行一次“压缩”操作:对于道 i,如果其对应的 BeFrontier$[i]$ 标志为 1,则将其加入数组 Frontier ,以作为下一次扩展操作的种子点。对 BeFrontier 进行“压缩”操作,可以减少下一次迭代中需要执行扩展操作所需的 CUDA 线程数量, 以避免不必要的计算。

算法 3 CUDAHorizonAdjuster(f)

```
Tracked[f] ← snap(f, Depth[f], Threshold[f])
if Tracked[f] = 1 then
  BeFrontier[f] ← 1
else
  BeFrontier[f] ← 0
end if
```

数据点的拾取过程如算法 4 所示,sanp 操作根据同相轴的相位类型来拾取数据点:同相轴的相位类型包括波峰(MAX)、波谷(MIN)、过零点(ZEROCROSS)三种,这三种类型的拾取操作分别是由 FindMax 、FindMin 和 FindZeroCross 完成的。

下面以 FindMax 操作为例进行介绍,另外两种操作的实现类似。FindMax 的伪代码如算法 5 所示。其中,depth 和 threshold 为扩展阶段得到的 Depth$[f]$ 和 Threshold$[f]$,SchRange 为偏离 depth 的最大深度偏差

值,SchRange 取值可正可负,若 SchRange 为正,则表示从 depth 向下搜索,若 SchRange 为负,表示向上搜索。Depth 和 SchRange 构成了数据点拾取的深度约束条件,threshold 则将作为数据点拾取的振幅阈值约束条件。GetValue(f, d) 函数用来获得道 f 上深度为 d 的数据点的振幅值。FindMax 的工作流程如下:

在目标道 f 上从 depth 深度的数据点 d 开始遍历,对 d 执行以下操作:

(1) 判断 d 是否为局部极大点,如果 d 不是局部极大点,则转(4)。

(2) 利用插值的方法计算出 d 附近的准确的极大点 max。

(3) 如果 max 的振幅满足阈值条件(对于 MAX 而言是大于阈值,对于 MIN 而言是小于阈值),则 max 即为满足约束条件的数据点,将 max 的深度值和振幅值分别写入记录层位面追踪结果的数组 ResultPos 和 ResultVal,返回 true,否则转(4)。

(4) 选择该目标道上下一个深度的数据点。如果该数据点的深度超过搜索的范围,则返回 false,表示未找到满足约束条件的数据点。否则更新 d 的值,转(1)。

算法 4 snap(f, depth, threshold)

```
found ← 0
if EventType = MAX then
  found = FindMax(f, depth, threshold)
else if EventType = MIN then
  found = FindMin(f, depth, threshold)
else if EventType = ZEROCROSS then
  found = FindZeroCross(f, depth)
end if
return found
```

算法 5 FindMax(f, depth, threshold)

```
for all d = depth to (depth + SchRange) do
  if IsMax(d) then
    max = FindNearestMax(d)
    if GetValue(f, max) ≥ theshold then
      ResultPos[f] = d
      ResultVal[f] = GetValue(f, max)
      return 1
    end if
  end if
end for
return 0
```

3　实验结果与分析

文中实验的硬件环境如下:CPU 为 Intel Core 2 Duo E6550@2.33 GHz,内存为 2 GB,显卡为 GeForce 9800GT。

在实验中,用 C++ 重新实现了开源地震解释软件 OpendTect [9] 的同相轴自动拾取层位面追踪算法,以作为文中算法的比较。该 CPU 版本实现对 OpendTect 原有的实现进行了改进,简化了数据结构,并除去了不必要的运算。

文中使用的实验数据为 Netherlands Offshore F3 Block,该数据总共有 651×951 道,每道有 463 个数据点。在实验中从 F3 数据体中选择不同大小的数据块,分别运行 GPU 版本和 CPU 版本的层位面追踪算法,记录它们的运行时间,得到图 1。表 1 显示了文中 GPU 算法和 CPU 算法在不同大小(即不同道数)的数据集上所用的时间。可见,CPU 版本算法随着数据量的增大,计算时间迅速增加,而 GPU 版本算法的计算时间随数据量的增大基本上呈线性增长趋势,而且随着数据量的增大,GPU 算法较 CPU 算法的加速也越明显。

表 1　GPU 算法和 CPU 算法所用时间比较

时间(ms)	200×200	300×300	400×400	600×500	600×800	651×951
GPU 算法	155	236	406	701	1039	1387
CPU 算法	198	489	1072	2632	5007	7441

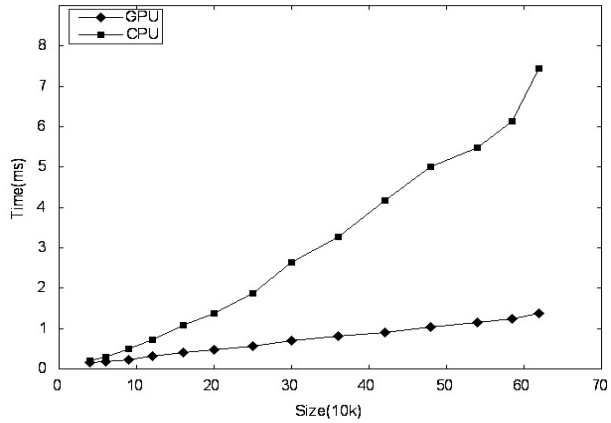

图 1　GPU 算法和 CPU 算法的执行时间曲线

图 2 显示了文中算法在 F3 数据体中追踪出来的完整结果(651×951 道)。对 F3 数据体中的所有道进行层位面追踪时,CPU 层位面追踪算法耗时 7441ms,GPU 算法耗时 1387ms。可见,与 CPU 版本的层位面自动追踪算法相比,文中算法的计算速度了有了明显的提升。

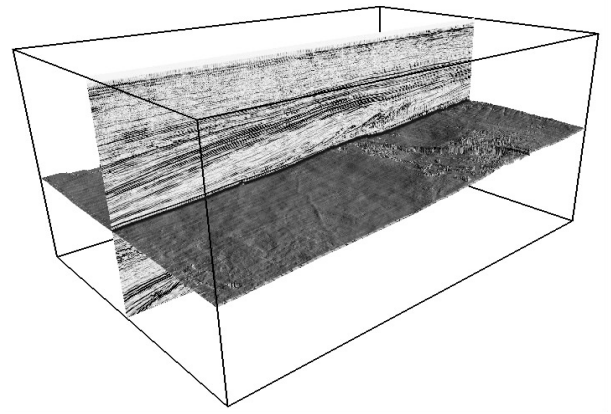

图 2　文中算法追踪出的层位面

(下转第 8 页)

参考文献:

[1] Singhal M. Issues and approaches to design of real-time database systems[J]. ACM SIGMOD Rec. ,1988,17(1):19-33.

[2] Cappellini V. A study on data-compression techniques[C]// European Space Research Organisation. [s. l.]:[s. n.], 1974.

[3] Nelson M, Gailly Jean-Loup. The data compression book [M]. 2nd ed. New York:MIS Press,1995.

[4] Iyer B R, Wilhite D. Data compression support in databases [C]//Proceedings of the 20th International Conference on Very Large Data Bases. [s. l.]:[s. n.],1994:695-704.

[5] Salomon D. Data compression: the complete reference[M]. New York:Springer-Verlag New York, Inc. ,1997.

[6] Roth M A, van Horn S J. Database compression[J]. ACM SIGMOD Rec. ,1993,22(3):31-39.

[7] Ehrman L. Analysis of some redundancy removal bandwidth compression techniques[J]. Proceedings of the IEEE,1967, 55(3):278-287.

[8] Hale J C, Sellars H L. Historical data recording for process computers[J]. CEP,1981,77(11):38-43.

[9] Bristol E H. Swinging door trending: adaptive trend recording [C]//ISA National Conf. Proc. . [s. l.]:[s. n.],1990:749-754.

[10] Kennedy J P. Data treatment and applications future of the desktop[C]//Proceeding of Foundations of Computer-aided Process Operations. [s. l.]:[s. n.],1993:21-43.

[11] Mah R S H. Process trending with piecewise linear smoothing [J]. Computers & Chemical Engineering,1995,19(2):129-137.

[12] 陆哲明. 矢量量化编码算法及应用研究[D]. 哈尔滨:哈尔滨工业大学,2001.

[13] 万 刚,朱长青. 多进制小波及其在DEM数据有损压缩中的应用[J]. 测绘学报,1999,28(1):36-40.

[14] 田宝凤,徐抒岩,孙荣春,等. 一种适合星上应用的遥感图像有损压缩算法[J]. 光学精密工程,2006,14(4):725-730.

[15] 肖振国,田 心. 医学图像无损压缩与有损压缩技术的进展[J]. 国外医学(生物医学工程分册),2002,25(2):55-58.

[16] Kortman C M. Redundancy reduction-a practical method of data compression[J]. Proceedings of the IEEE,1967,55(3): 253-263.

[17] James P A. Data compression for process historians[C]// CAST Communications. [s. l.]:[s. n.],1996.

[18] 祝 君,林庆农,徐造林,等. 实时历史数据库中压缩技术的并行化研究[J]. 计算机技术与发展,2010,20(7):36-39.

[19] 陈建英,刘心松. 基于大规模分布式副本定位的分级索引压缩机制[J]. 电子科技大学学报,2011,40(4):554-558.

(上接第4页)

4 结束语

文中提出了一种基于CUDA的三维层位面自动追踪算法,跟原有的CPU版本的层位面自动追踪算法相比,计算速度有了明显的提升。但是由于文中采用的层位面自动追踪算法是一个不断迭代的过程,每一次迭代都需要进行显存和内存之间的数据拷贝。同时由于再执行调整操作时,不同CUDA线程的计算量并不完全相同,这就导致了thread divergence[12],这也限制了文中算法的速度。如何解决这两个问题是下一步工作的重点。

参考文献:

[1] 陆基孟. 地震勘探原理(上册)[M]. 东营:石油大学出版社,1993.

[2] Faraklioti M, Petrou M. Horizon picking in 3D seismic data volumes[J]. Machine Vision and Applications,2004,15(4): 216-219.

[3] 李雪峰,阎建国,赵 州,等. 利用相干属性剖面特征进行层位解释[J]. 物探化计算技术,2011,33(2):134-139.

[4] Bahorich M, Farmer S. The coherence cube[J]. The Leading Edge,1995,14(10):1053-1058.

[5] NVIDIA Corporation. NVIDIA CUDA C Programming Guide, Version 4.0[EB/OL]. 2011. http://developer. nvidia. com/nvidia-gpu-computing-documentation.

[6] Sanders J, Kandrot E. CUDA by Example: An Introduction to General-purpose GPU Programming[M]. 北京:清华大学出版社,2010.

[7] Deschizeaux B, Blanc Jean-Yves. Imaging Earth's Subsurface Using CUDA[C]//GPU Gems 3. [s. l.]: Addison Wesley Professional,2007.

[8] Keskes N, Zaccagnino P, Rether D, et al. Automatic extraction of 3-D seismic horizons[C]//Annual Meeting Expanded Abstracts. [s. l.]:SEG,1983:557-559.

[9] dGB Earth Sciences. OpendTect User Documentation Version 4.0[EB/OL]. 2009. http://opendtect. org/rel/doc/User/base/.

[10] Harris M. Parallel Prefix Sum with CUDA[C]//GPU Gems 3. [s. l.]:Addison Wesley Professional,2007.

[11] Harish P, Narayanan P J. Accelerating large graph algorithms on the GPU using CUDA[C]//High performance computing-HIPC 2007, lecture notes in computer science. [s. l.]:[s. n.],2007:197-208.

[12] Kirk D B, Hwu W W. Programming Massively Parallel Processors[M]. [s. l.]:Morgan Kaufmann, 2010:96-103.

基于 CUDA 的地震数据层位面自动追踪算法

作者： 魏强， 唐杰
作者单位： 南京大学 计算机科学与技术系,江苏 南京 210093
刊名： 计算机技术与发展
英文刊名： Computer Technology and Development
年，卷(期)： 2012(9)