文章编号：2096-1472(2018)-12-19-03
DOI:10.19644/j.cnki.issn2096-1472.2018.12.006

基于GPU的叠前逆时偏移混合粒度数据分割与存储优化

韩 菲[1],李 炜[2]
(1.联想(北京)有限公司，北京 100094；
2.联想(北京)信息技术有限公司，北京 100094)

摘 要：为提高叠前逆时偏移计算效率，本文采用MPI+CUDA混合粒度相结合的并行模式，对地震数据进行数据分割，合理划分并行任务。总结出MPI+CUDA并行编程模型，提出叠前逆时偏移的混合粒度并行算法。根据CUDA特有的存储方式，对叠前逆时偏移算法提出存储优化方案，更高效的利用GPU上各类存储器，以进一步降低数据访问所造成的时间延迟。

关键词：图形处理器；叠前逆时偏移；混合粒度；并行计算；存储优化
中图分类号：TP391 **文献标识码**：A

The Hybrid Granularity Data Segmentation and Storage Optimization of Prestack Reverse-Time Migration Based on GPU

HAN Fei[1],LI Wei[2]
(1.*Lenovo Beijing Co.LTD.,Beijing* 100094,*China*;
2.*Lenovo Beijing Information Technology Co.LTD.,Beijing* 100094,*China*)

Abstract: To improve the computational efficiency of prestack reverse-time migration,this paper adopts the MPI + CUDA parallel model to divide seismic data and parallel tasks.The MPI + CUDA parallel programming model is summarized and the hybrid granularity parallel algorithm of prestack reverse-time migration is proposed.Based on the special storage model of CUDA,we propose the storage optimization scheme for prestack reverse-time migration algorithm so as to reduce time delay caused by the data access with higher involvements of all kinds of memories on GPU.

Keywords:GPU;prestack RTM;hybrid granularity;parallel computing;storage optimization

1 引言(Introduction)

高性能计算技术在飞速发展，从微处理器由单核发展到多核，从并行机的出现到集群计算技术的飞速发展，无不证明计算机技术的飞速进步。当拥有多核处理器的GPU出现[1]，代替CPU处理大规模并行问题时，更是将高性能计算推向了一个新的高峰。目前，GPU已经广泛应用于各个领域，并大大提高了计算密集问题的计算效率。

GPU在石油地球物理勘探领域有着广泛应用。美国休斯顿的Headwave公司专门从事地学数据分析[2]，充分利用GPU的并行计算潜力，支撑实时工作流程、实时可视化和实时计算，支持叠前地震数据的分析与解释。目前，GPU已在叠前逆时偏移中逐渐展开应用。因逆时偏移计算量且存储量巨大，无法适应工业生产的需要，一直没有在工业界得到广泛应用，如何提高逆时偏移的计算效率成为逆时偏移能否大规模应用的重要环节[3]。如何合理地划分计算任务，对计算数据进行合理存储，提高存储速度，成为我们首要研究的内容。本文采用粗、细粒度混合的数据分割方式划分并行任务，给出叠前逆时偏移算法下MPI+CUDA的混合计算模型，研究了基于GPU叠前逆时偏移的并行算法。根据CUDA特有的存储方式，对逆时偏移算法的存储提出优化方案。

2 叠前逆时偏移原理(The principle of prestack reverse time migration)

叠前逆时深度偏移求解两次波动方程，即从源点以时间正方向正演出空间所有点的波场及从检波点位置以时间逆序推出所有空间反射点的波场；利用相应成像条件对两个波场进行成像。逆时偏移是全波场的双程波动方程偏移方法[4]。可以对复杂速度体成像，不受成像倾角限制(可达到180°)；能对所有波动现象(如反射波、折射波、绕射波、回转波、棱柱波和多次波等)进行成像。此过程中波的传播是相对完整的，因而所用偏移速度模型无论如何复杂都不需要平滑处理，但对于常规Kirchhoff偏移是很难做到的。

首先，给出该方法的具体实施流程：(1)以时间正方向正

演计算震源波场；(2)以时间逆方向逆推接收点波场；(3)利用成像条件，对两波场成像；(4)所有炮集成像结果叠加得到最后成像结果。

震源波场正演计算采用的声波方程表示为式(1)

$$\left(\frac{1}{v^2}\frac{\partial^2}{\partial t^2}-\nabla^2\right)p_F(\vec{x};t)=\delta(\vec{x}-\vec{x}_s)f(t) \tag{1}$$

式中v代表速度，p代表波场，$f(t)$是震源函数。为数值计算方便，把震源函数作为边界条件处理，因此式(1)就可转换成等价的式(2)

$$\begin{cases}\left(\frac{1}{v^2}\frac{\partial^2}{\partial t^2}-\nabla^2\right)p_F(\vec{x};t)=0\\ p_F(x,y,z=0;t)=\delta(\vec{x}-\vec{x}_s)\int_0^t f(s)ds\end{cases} \tag{2}$$

式中p_F为由式(2)所求的正演波场。

接收点波场的计算采用式(3)，与式(2)不同之处是修改了边界条件：

$$\begin{cases}\left(\frac{1}{v^2}\frac{\partial^2}{\partial t^2}-\nabla^2\right)p_B(\vec{x};t)=0\\ p_B(x,y,z=0;t)=Q(x,y;x_s,y_s;t)\end{cases} \tag{3}$$

式中p_B为所求的接收点波场，Q为地震记录。

对于式(2)和式(3)的数值求解方法较多，如有限差法分、有限元和伪谱法等，每种方法都有各自的优缺点。本文采用的数值求解方法是时间方向为四阶差分精度的伪谱法。通过数值计算得到震源波场p_F和接收点波场p_B后，然后利用相应成像条件对两个波场进行成像。那么，不同成像条件的应用，将直接影响成像的振幅、分辨率及其保幅性。所以，成像条件的选择比较关键，对保幅性的叠前深度偏移更是至关重要。

互相关成像条件式(4)，由正演得到的震源波场$S(x,y,z,t)$和由逆时延拓得到的接收点波场$R(x,y,z,t)$，利用互相关成像：

$$image(x,y,z)=\sum_{time}S(x,y,z,t)R(x,y,z,t) \tag{4}$$

3 CPU/GPU协同架构(CPU/GPU collaborative architecture)

CUDA是一种将GPU作为数据并行计算设备的软硬件体系，目前已广泛应用。在运算过程中GPU(设备)是作为CPU(主机)的协处理器来执行操作的，主机和设备均有自己的存储器。CPU-GPU并行编程模型如图1所示[5]。

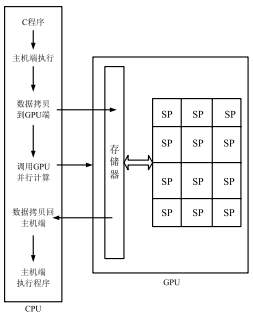

图1 CPU-GPU并行编程模型

Fig.1 CPU-GPU parallel programming model

4 混合粒度数据并行算法(Hybrid-grained data parallel algorithm)

粒度是对一个并行计算任务中计算量大小和通信量大小比值的度量。将问题分解为多个子问题，根据并行计算任务的大小，可以分为粗粒度并行、细粒度并行和中粒度并行。本文采取粗粒度并行与细粒度并行两种并行方式设计算法。

4.1 粗粒度数据并行

4.1.1 粗粒度数据分割

粗粒度是将一个任务分解为含有较多计算量的多个子问题，这些子问题可以独立地并行解决问题。本文采用MPI粗粒度并行方式对地震数据进行数据分割。首先由主节点(Master node)根据有效节点数N_n和地震数据总炮数N_s进行节点间作业任务分配。为了提高读取数据的速度，采用多节点并行读取数据，能大大提高数据输入输出效率。每个节点单炮偏移完后，返回主节点判断是否所有炮集数据都计算处理完毕，若没完成进入下一次循环，直至所有炮集资料都偏移完为止。粗粒度数据分割如图2所示。

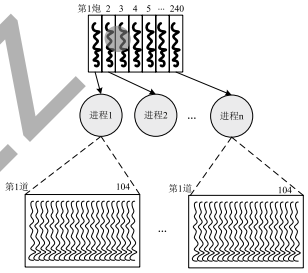

图2 MPI粗粒度数据分割

Fig.2 MPI Coarse-grained data segmentation

4.1.2 粗粒度数据并行

利用多GPUs和多核CPU协同加速计算逆时偏移，主要思想：因逆时偏移是按照炮集一炮一炮进行偏移，采用MPI启动多进程，每个进程读取一炮地震数据，将一炮地震数据发送到一块GPU上进行运算，在GPU上启动多线程并行执行一炮地震数据的偏移过程。通过调度实现在多GPUs上并行，将读取文件和其他步骤设计成串行过程。充分利用GPU高效的浮点数处理能力和多核CPU良好的任务分配和调度能力。多GPUs算法流程如图3所示。

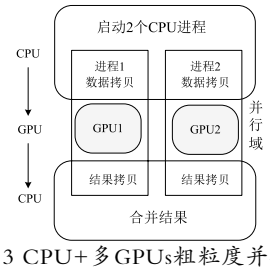

图3 CPU+多GPUs粗粒度并行

Fig.3 CPU+multi-GPUs coarse grained parallelism

4.2 细粒度数据并行

4.2.1 细粒度数据分割

细粒度是一个任务划分成包含较短的程序段和较小的计算量的并行任务，粒度越小，越可开发更多的并行性，提高

并行度，这是有利的方面。但粒度越小，通信次数和通信量就相对增多，增加了额外开销。GPU可以同时开启成百上千个线程，拥有强大的数据处理能力，专用于解决数据并行计算的问题，具有极高的计算密度(数学运算与存储器运算的比率)，是典型的细粒度并行编程模型。

本文采用GPU上细粒度数据分割方式，将MPI一个进程读取的一炮地震数据通过主机发送到GPU上。每炮地震数据有104道地震记录，每道地震记录发送到GPU的一个线程上，每个线程计算一道地震数据。线程细粒度数据分割如图4所示。

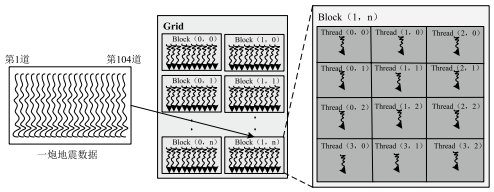

图4 线程级细粒度地震数据分割

Fig.4 Thread-level fine-grained seismic data segmentation

4.2.2 细粒度多线程并行

在GPU上启动多线程，将一炮地震数据中104道地震数据映射到线程中，以取代正演和逆推过程中偏移一炮地震数据从第一道循环到第104道，从第1个采样点到1024采样点的两层循环。将计算量均匀地加载到各线程，提高算法的并行度，使算术逻辑单元尽可能均匀地分配到计算任务，是提高GPU计算效率的举措之一。

4.3 叠前逆时偏移MPI+CUDA混合粒度并行算法

结合粗粒度与细粒度并行的优点，考虑采用MPI+CUDA多粒度混合编程模型，节点间通过MPI消息传递机制，实现粗粒度并行；节点内利用GPU强大的并行数据处理能力，使用CUDA架构启用多线程机制，利用共享内存实现高效快速细粒度的数据并行和线程并行。混合粒度编程模型如图5所示。MPI+CUDA混合编程模型能充分发挥集群节点间分布式存储和节点内共享存储的优势，既可以发挥每个节点的巨大计算能力，又可以充分利用集群的可扩展性，使并行效率显著提高，为CPU+GPU混合架构做大型计算提供了一种有效的并行策略。

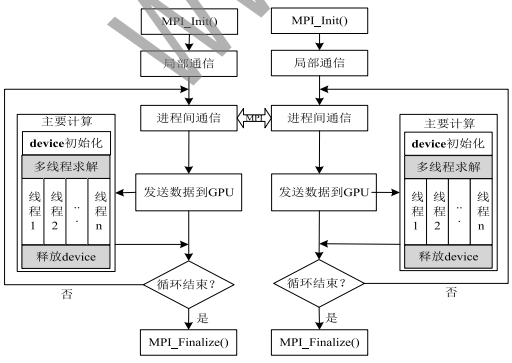

图5 MPI+CUDA混合粒度并行计算编程模型

Fig.5 MPI+CUDA mixed granularity parallel computing programming model

叠前逆时偏移算法的主要部分：震源波场的正传，检波点波场的反传，以及相关成像条件均是CPU串行计算最为耗时的部分。针对共炮域的逆时偏移而言，本文采取混合粒度的并行方式，以单个炮集为粗粒度并行，通过GPU并行计算的方法，将每个炮集包含的104道地震数据发送到GPU中执行线程级细粒度并行。在GPU上，先将速度数据、一炮的地震数据和激发点的初始波场值由内部存储器(内存)传至设备存储器(显存)，这样可以避免由多次重复对内部存储器的数据读写所引起的时间延迟。然后把GPU的多核处理器划分为相应个数的计算块(block)，同时每个计算块又可以划分为若干个线程(thread)，利用GPU的多线程实现大规模的并行计算。这样就可以使得在计算过程中涉及的数据读写和运算均在设备存储器和图形处理器(GPU)中进行，将波场延拓及相关成像的计算并行化，达到提高计算效率的目的。最后再将数据传回内部存储器，通过CPU进行结果的I/O操作。

5 逆时偏移存储优化(Reverse time migration storage optimization)

5.1 利用高速存储器进行优化计算

寄存器是多处理器上最快的片上存储器。叠前逆时偏移并行算法内核函数中的变量在允许情况下都定义为寄存器型。每个SM具有32位寄存器，如果内核编译时使用了超过执行配置允许数量的寄存器，会造成内核由于寄存器不足而无法启动。叠前逆时偏移并行算法中每个SM使用寄存器小于32kB，没有超限。

CUDA提供了具有高速读写访问并行数据的共享存储器，其速度在无存储体冲突时等同于寄存器。共享存储器是可以被同一块中的所有线程访问的可读写存储器，应用程序可以利用它来最小化对DRAM的过度提取和巡回，从而降低对DRAM存储器带宽的依赖程度。并行算法内核函数中除了部分计算参数和中间型数据运算时需要共享型外，把用于接收从设备存储器传入的目标区和搜索区的地震数据也定义为共享型。由于每个SM上只有48kB共享存储器，一个SM可以同时执行多个线程块，片内的存储资源则会被分配给这些并发的线程块中。根据每个SM的资源上限，计算分发单元就可以知道最多能在一个SM上分配多少个block。若在计算相关系数的内核函数kernel中每个block中有104个线程，每个block使用49152B共享存储器，每个线程使用四个寄存器，那么可以有：每个block使用的共享存储器：49152Byte；每个block使用的寄存器数量：4*104=416；每个block中的warp数量：ceil(416/32)=13。

根据每个block使用的资源，就可以计算出kernel中由每个因素限制的最大活动块数量：每个SM中的最大活动块数量：8；由共享存储器数量限制的活动块数量：floor(393216/49152)=8；由warp数量限制的活动块数量：floor(32/13)=2。

5.2 利用常数存储器进行优化

对当前硬件来说，一个half-warp中的线程访问常数存储

(下转第11页)