doi: 10.11720/wtyht.2016.1.32
李午阳,张健,林巍.基于 CUDA 的 GPU 并行优化重力三维反演[J].物探与化探,2016,40(1):179-184.http://doi.org/10.11720/wtyht.2016.1.32
Li W Y,Zhang J,Lin W.Regularized inversion of 3D gravity data: a new GPU parallelized method based on CUDA [J].Geophysical and Geochemical Exploration,2016,40(1):179-184.http://doi.org/10.11720/wtyht.2016.1.32

基于 CUDA 的 GPU 并行优化重力三维反演

李午阳[1,2],张健[1,2],林巍[3]
(1.中国科学院大学 地球科学学院,北京 100049; 2.中国科学院 计算与地球动力学实验室,北京 100049; 3.犹他大学 地质与地球物理系,美国 盐湖城 84112)

摘 要: 笔者介绍了一种在 PGI Fortran 平台上开发的重力三维 GPU 并行反演算法。该方法采用重加权正则化共轭梯度算法(Re-Weight Regularized Conjugate Gradient),可以在具有 NVIDIA 显卡的个人计算机上使用 CUDA 进行并行计算,无需借助工作站即可实现几十至上百倍的计算加速,提供稳定可信的反演结果。并对可视化操作系统进行了优化,实现了在高端计算机系统上亿网格点的反演计算,同时在中、低端计算机也可以实现加速。模型计算结果表明,该算法是一种高效且可靠的重力三维反演并行方法。

关键词: 重力;三维反演;CUDA;GPU;并行计算;共轭梯度法
中图分类号: P631 **文献标识码**: A **文章编号**: 1000-8918(2016)01-0179-06

重力勘探由于其成本较低、施工方法方便等,被广泛应用于大尺度的地质异常体勘查、大范围找矿普查、以及小比例尺密度三维地质建模等工作中。目前常用的反演方法有两种,2.5 维联合 3 维界面反演[1-2]和三维物性反演。通过对地下介质进行三维网格剖分,反演每个网格单元的物性,从而得到地质异常体的物性分布与形态特征[3]。

对于复杂结构目标体的三维反演,由于受到数据计算与存储量的限制,主要采用的是三维物性反演。即便如此,重力场计算的三维正演模拟和反演中依旧面临着很大的数据处理量,使用传统的线性反演,往往数百万个网格点的计算就会超过个人计算机的内存处理范围。由于计算机内存与 CPU 计算效率的瓶颈,很大程度上限制了数据处理解释工作的品质。为了解决这个问题,近十几年来,并行计算成为了主流的研究方向。Chen 等[4-5]采用了基于 C 语言的 CUDA 代码实现了重力的三维正演成像;林巍等[6]利用 GPU 遗传算法实现了海洋重力三维反演;Cuma 等[7-8]实现了基于 CPU 的 OpenMP 算法加速与基于 GPU 的 OpenACC 算法加速,提供了在服务器等高性能计算机集群上实现重力反演计算的方法。从硬件结构上来看,相较于目前仅有数个、数十个核的 CPU,拥有数百甚至数千个核的 GPU 在处理大数据量的简单运算,如重力反演大矩阵运算上,有着天然的优势。

笔者在 PGI Fortran 平台上采用传统 Fortran 语言与 CUDA 语言混编,将重加权正则化共轭梯度(Re-weighting Regularized Conjugate Gradient, RRCG)三维重力场反演算法进行重组,使其可以在个人台式计算机与笔记本计算机上使用 GPU 进行并行计算,正则化的使用使反演结果稳定可信、具有物理意义;放弃存储敏感度矩阵 $\boldsymbol{A}$,从而使得计算效率与数据处理量同时得到提高;引入了线程多次调用与敏感度范围(footprint)搜索,实现了存储扩容与反演加速。为了验证程序的正确性,我们采用相同算法的 MATLAB 传统代码进行了比对测试。该方法可以有效的在个人计算机上解决上亿网格点的反演计算问题,为解决大数据量的三维重力反演提供了新的方法思路。

1 正演与反演

1.1 正演算法

虽然对于重力分量体积分的解析解是可以直接求取的,但为考虑并行计算的效率,笔者仍采用常用

收稿日期: 2015-11-07
基金项目: 国家自然科学基金项目(41574074、41430319)

的离散化单点积分:将地下介质划分成均匀的立方体网格,并假设网格内部密度均匀统一,经过网格离散化以后,在重力测量中常用的垂直分量计算公式可以表示为

$$g_z(\boldsymbol{r}) = \gamma \sum_{k=1}^{N_m} \rho_k \frac{z_k - z}{|\boldsymbol{r} - \boldsymbol{r}'|^3} \Delta x \Delta y \Delta z \tag{1}$$

离散形式。其中:γ 表示万有引力常数;N_m 表示立方体网格总数;k 表示网格点编号;矢量 $\boldsymbol{r}' = (x', y', z')$,$\boldsymbol{r} = (x, y, z)$ 分别表示观测点和源网格中心点的位置;z 和 z' 分别表示观测点和源中心点在垂直方向上的位置;Δx, Δy, Δz 分别表示划分立方体网格沿相对坐标轴三个方向的长度;ρ 表示立方体网格的密度。

在这种条件下,求观测点重力异常值的过程可以看成是一个线性算子 $\boldsymbol{A}$ 与密度分布矢量 $\boldsymbol{m}$ 之间的运算

$$\boldsymbol{d} = \boldsymbol{A}\boldsymbol{m}, \tag{2}$$

其中:$\boldsymbol{d}$ 表示为一个长度为 N_{data} 的观测点空间分布矢量,$\boldsymbol{m}$ 表示为一个长度为 M_{cell} 的空间密度分布矢量,$\boldsymbol{A}$ 表示重力垂直分量的线性算子,在这样的线性问题下,$\boldsymbol{A}$ 也是反演问题中的敏感度矩阵。

1.2 反演算法

考虑到并行计算速度,笔者在反演中采用的是重加权的正则化共轭梯度(RRCG)算法,Tikhonov 正则化的引入,可以压制由于数据噪声与多解性引起的反演结果不稳定与物理意义不明确[9-10],而不同的加权因子可以在平滑解与有明显边界的解之间进行平衡,重加权(Re-weighting)可以在迭代过程中动态调整正则化因子 α 的范围,可以在保证数据品质的前提下加快收敛速度。笔者仅介绍与并行计算相关的算法流程,对于加权因子与正则化因子的选择与更新方式不再介绍,可详见文献[9]。

RRCG 反演算法求取的是参数泛函(Parameter Functional)的极小值,其完整形式可以表示为

$$P^{\alpha}(\boldsymbol{m}) = (\boldsymbol{W}_{\text{d}}\boldsymbol{A}\boldsymbol{m} - \boldsymbol{W}_d\boldsymbol{d})^{\text{T}}(\boldsymbol{W}_d\boldsymbol{A}\boldsymbol{m} - \boldsymbol{W}_d\boldsymbol{d}) + \alpha(\boldsymbol{W}_e\boldsymbol{W}_m\boldsymbol{m} - \boldsymbol{W}_e\boldsymbol{W}_m\boldsymbol{m}_{\text{apr}})^{\text{T}}(\boldsymbol{W}_e\boldsymbol{W}_m\boldsymbol{m} - \boldsymbol{W}_e\boldsymbol{W}_m\boldsymbol{m}_{\text{apr}}), \tag{3}$$

其中:$\boldsymbol{d}$ 为观测数据矢量;$\boldsymbol{m}$ 为模型矢量;$\boldsymbol{A}$ 为正演算子;$\boldsymbol{m}_{\text{apr}}$ 是先验模型矢量;α 是正则化参数,用来平衡右式第一项误差泛函(misfit functional)与第二项稳定泛函(stabilizer functional);$\boldsymbol{W}_d$、$\boldsymbol{W}_m$ 分别是数据和模型加权矩阵,$\boldsymbol{W}_e$ 是聚焦加权矩阵。在位场反演中,不对观测数据加权,即 $\boldsymbol{W}_d$ 为单位矩阵,而 $\boldsymbol{W}_m$ 与 $\boldsymbol{W}_e$ 皆为对角阵,针对不同问题有不同的调整方法。

共轭梯度法的是计算是用迭代计算替代求逆矩阵计算,迭代计算的每一步的计算流程如下[11]

$$\boldsymbol{r}_n = \boldsymbol{A}\boldsymbol{m} - \boldsymbol{d}, \tag{4}$$

$$\boldsymbol{l}_n^{\alpha} = \boldsymbol{F}_{\boldsymbol{m}_n}^{T}\boldsymbol{r}_n + \alpha_n(\boldsymbol{W}_m\boldsymbol{W}_e)^2(\boldsymbol{m}_n - \boldsymbol{m}_{\text{apr}}), \tag{5}$$

$$\beta_n^{\alpha_n} = \frac{\|\boldsymbol{l}_n^{\alpha_n}\|^2}{\|\boldsymbol{l}_{n-1}^{\alpha_{n-1}}\|^2},\ \tilde{\boldsymbol{l}}_n^{\alpha_n} = \boldsymbol{l}_n^{\alpha_n} + \beta_n^{\alpha_n}\tilde{\boldsymbol{l}}_{n-1}^{\alpha_{n-1}},\ \tilde{\boldsymbol{l}}_0^{\alpha_0} = \boldsymbol{l}_0^{\alpha_0}, \tag{6}$$

$$k_n^{\alpha_n} = \frac{(\tilde{\boldsymbol{l}}_n^{\alpha_n}, \boldsymbol{l}_n^{\alpha_n})}{\|\boldsymbol{F}_{\boldsymbol{m}_n}\tilde{\boldsymbol{l}}_n^{\alpha_n} + \alpha\boldsymbol{W}_m\boldsymbol{W}_e\tilde{\boldsymbol{l}}_n^{\alpha_n}\|^2},\ \boldsymbol{m}_{n+1} = \boldsymbol{m}_n - k_n^{\alpha_n}\tilde{\boldsymbol{l}}_n^{\alpha_n}, \tag{7}$$

其中:$\boldsymbol{r}_n$ 表示剩余误差矢量;n 表示迭代次数;$\boldsymbol{l}_n$ 表示梯度法迭代方向矢量,$\tilde{\boldsymbol{l}}_n$ 表示共轭梯度法迭代方向矢量;$\boldsymbol{F}$ 表示 Frechet 导数矩阵;在线性反演问题中 $\boldsymbol{F} = \boldsymbol{A}$;$k_n$ 表示每一次迭代的迭代步长;α_n 是每一步迭代的正则化参数。

传统的串行 RRCG 反演中需要计算、存储敏感度矩阵 $\boldsymbol{A}$ 并反复读取,顺序计算每一个迭代步骤,直到迭代满足次数条件或误差条件从而停止。$\boldsymbol{A}$ 矩阵是一个与数据及模型尺度相乘有关的大矩阵,在大型三维反演中存储和读取耗时且困难,成为限制常规 RRCG 线性反演的重要因素。

2 GPU 算法设计

2.1 GPU 与 CUDA

GPU(Graphics Processing Unit),即图形处理单元,计算方式高度并行,计算能力强大,被早期的科研工作者用于提高通用计算的计算速度。而近十年来,由于 CUDA 的出现,才使得 GPU 成为一种不需要掌握图形计算基础就可以使用的通用并行计算工具[12]。CUDA(Compute Unified Device Architecture)是由 NVIDIA 提供的通用 GPU 编程语言架构,目前可以支持在 C、Fortran、Matlab 等多种语言平台下使用 CUDA 语句调用 GPU 计算单元进行并行计算。

CUDA 是最适合应用在位场线性三维反演中的并行计算手段之一。首先,采用 RRCG 反演方法,其流程是用迭代算法,每个计算步骤都可以简化为相互独立的矩阵与矢量进行简单运算,为并行计算提供了条件;其次,与传统的 CPU 串行或并行相比,GPU 的计算核心数量有绝对领先的优势,相较于目前高端个人计算机拥有的 16 个或 32 个计算核心,个人显卡即可拥有 400~600 个,甚至最高超过 2000 个 GPU 计算核心;最后,相比于最近发布的另一种 GPU 计算标准 OpenACC,虽然 CUDA 需要更多的对

计算过程的控制和分配,即更多的对于常规反演代码的改编,但同时留有更多对显存地控制和深度开发的余地。

与此同时,在拥有良好的算法结构与计算控制的情况下,与 OpenACC 相比,CUDA 的计算效率可以最多提高一倍以上[13]。因此,笔者采用了基于 PGI Fortran 的 CUDA 作为并行算法的开发平台。

2.2　RRCG 反演的并行设计

RRCG 反演迭代流程中的每一步计算结果皆为标量或矢量而非矩阵,在实际运算中,标量与矢量相互间的运算速度非常快,可交由 CPU 计算,而需要并行的部分是与敏感度矩阵 $\boldsymbol{A}$ 及其转置有关的矩阵运算。可以按照计算方式将并行部分分为两类:数据尺度 N_{data} 的并行,即与 $\boldsymbol{A}$ 矩阵有关的矩阵运算,计算结果为数据长度的矢量 $\boldsymbol{I}_{\text{par}}$;模型尺度 M_{cell} 的并行,与 $\boldsymbol{A}$ 矩阵转置有关的矩阵运算,计算结果为模型尺度 M_{cell} 的矢量 $\boldsymbol{J}_{\text{par}}$。由此,可以将迭代公式改写为

$$\boldsymbol{r}_n = \boldsymbol{I}_{\text{par1}}[N_{\text{data}}] - \boldsymbol{d}, \tag{8}$$

$$\boldsymbol{l}_n^{\alpha} = \boldsymbol{J}_{\text{par1}}[\boldsymbol{M}_{\text{cell}}] + \alpha_{\text{n}}(\boldsymbol{W}_{\text{m}}\boldsymbol{W}_{\text{e}})^2(\boldsymbol{m}_n - \boldsymbol{m}_{\text{apr}}), \tag{9}$$

$$\beta_n^{\alpha_n} = \frac{\| \boldsymbol{l}_n^{\alpha_n} \|^2}{\| \boldsymbol{l}_{n-1}^{\alpha_{n-1}} \|^2},$$

$$\tilde{\boldsymbol{l}}_n^{\alpha_n} = \boldsymbol{l}_n^{\alpha_n} + \beta_n^{\alpha_n}\tilde{\boldsymbol{l}}_{n-1}^{\alpha_{n-1}},\ \tilde{\boldsymbol{l}}_0^{\alpha_0} = \boldsymbol{l}_0^{\alpha_0}, \tag{10}$$

$$k_n^{\alpha_n} = \frac{(\tilde{\boldsymbol{l}}_n^{\alpha_n}, \boldsymbol{l}_n^{\alpha_n})}{\| \boldsymbol{I}_{\text{par2}}[N_{\text{data}}] + \alpha \boldsymbol{W}_m \boldsymbol{W}_e \tilde{\boldsymbol{l}}_n^{\alpha_n} \|^2},$$

$$\boldsymbol{m}_{n+1} = \boldsymbol{m}_n - k_n^{\alpha_n}\tilde{\boldsymbol{l}}_n^{\alpha_n}, \tag{11}$$

其中:矢量 $\boldsymbol{I}_{\text{par1}}$、$\boldsymbol{I}_{\text{par2}}$ 与 $\boldsymbol{J}_{\text{par1}}$ 分别代表了上文所述的两类并行化的矩阵运算的结果。

将矩阵与矢量的乘积运算拆分,分配给 GPU 计算单元,让每个单元独立进行网格体积、观测点距离等计算,只输出结果矢量。通过这样的调整,无需像传统方法一样预先将庞大的敏感度矩阵 $\boldsymbol{A}$ 存入系统内存,大大降低了对系统内存的需求(图 1b、1c);且每次重新计算与 $\boldsymbol{A}$ 矩阵有关的矢量所消耗的时间要远远低于访问庞大矩阵所花费的时间,提高了计算效率。对于无需并行的部分,交由 CPU 进行串行计算,迭代计算流程如图 1a 所示。

2.3　计算效率

为验证并行计算的加速效率,将并行代码与采用相同反演算法的常规串行代码进行计算时间比较,比较代码的计算结果由犹他大学蔡红柱博士提供,编程平台为 MATLAB,重力垂直分量模拟测量点 5 000 个。计算数据在同一台中端笔记本计算机上获得,该机型拥有 16Gb 内存,CPU 是拥有 4 核 8 线程的 i7-4710,主频率为 2.50 GHz;显卡为 GeForce 860m,拥有640个GPU计算单元与4Gb显存,使用其中的512个单元,下文中所涉及的所有计算均采

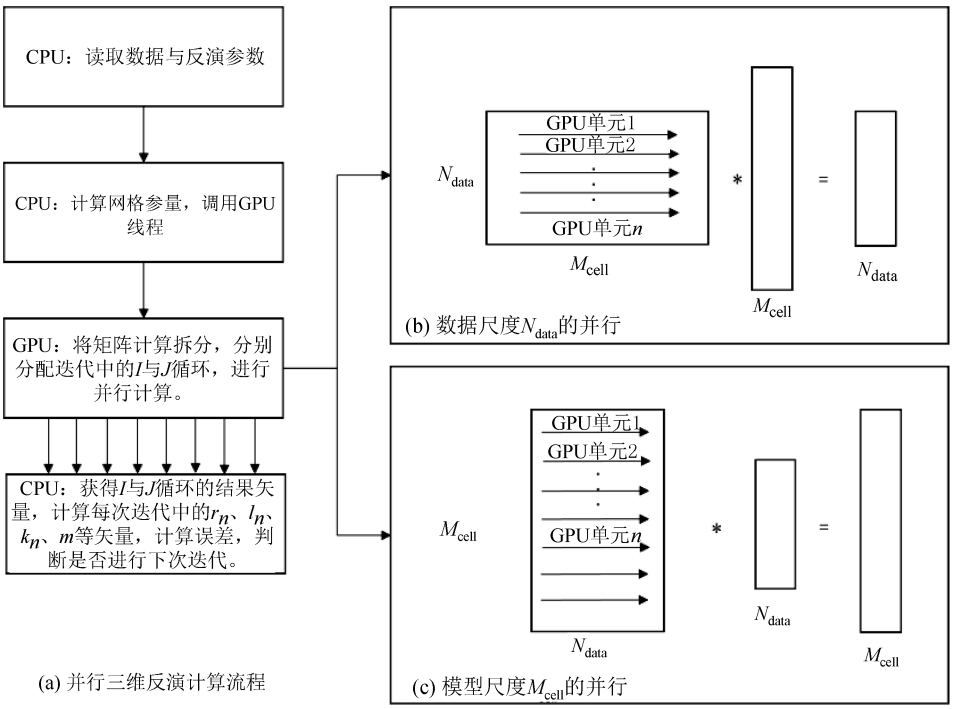


图 1　并行化 RRCG 三维反演计算流程

表 1 串行与并行代码 RRCG 反演单次迭代计算时间对比

网格数量	串行代码单次迭代时间/s	并行代码单次迭代时间/s
64000(40×40×40)	0.44	0.098
512000(80×80×80)	3.73	0.74
1000000(100×100×100)	196.13	1.47
2000000(100×100×200)	内存溢出	2.86

用相同配置。计算时间如表 1 所示。

从表中可以看出,随着网格数量的增加,传统串行方法中对敏感度矩阵 $\boldsymbol{A}$ 的计算与存储将会占用大量的计算时间与内存,当网格数量达到 100 万以上时,16 Gb 的计算机内存已经不足以支持存储 $\boldsymbol{A}$ 矩阵。相反,经过 GPU 并行的反演算法,由于矩阵运算已经分配到 GPU 计算单元中,只存储与 $\boldsymbol{A}$ 及其转置矩阵有关的矢量,即矢量 $\boldsymbol{I}$ 与 $\boldsymbol{J}$,从而克服了内存占用的瓶颈,迭代计算时间也仅仅表现出随网格数量增加而近似线性增加的关系。

综上,通过将重力三维反演直接并行化,已经可以实现在相同整机配置水平下 CPU 计算性能的几十甚至几百倍的加速。然而由于 GPU 使用本身的原因,单次迭代时间被限制在了 5 s 以内,从而网格点数量被限制在了 400 万左右。将介绍两种优化计算的方法,进一步提高计算网格上限与计算速度。

3 优化计算技术

3.1 线程多次调用

个人计算机常用的操作系统,无论是 Linux、Mac 或 windows,都拥有可视化桌面(Visual Windows)功能,当使用 CUDA 语言调用 GPU 线程进行工作时,单次运算的时间如果超过 5~15 s,为保护可视化界面的正常功能,并行计算将会被系统强制停止,这种问题称之为“调用瓶颈”。为此,采取 GPU 线程多次调用技术,尽可能的将 GPU 计算单元的单次时间使用压制在 5 s 以内。

具体做法是在 CPU 调用 GPU 线程之前,将单个 GPU 核心的计算量以 $\boldsymbol{M}_{\text{cell}}$ 的整数倍进行切割,减少 GPU 计算单元内部的 loop 循环大小,从而减少单次计算时间。在计算 $\boldsymbol{I}_{\text{par}}$ 矢量时,由于每次调用仅计算每一个数据点的部分结果,对计算结果进行累加即得到完整的迭代解。而计算 $\boldsymbol{J}_{\text{par}}$ 矢量时,则需要对每次调用的运算结果进行矢量拼接。

对多次调用优化后的代码进行了单次迭代时间测试,结果如表 2 所示,其中单次时间溢出表示单次线程计算时间超过调用瓶颈。结果表明,多次调用线程可以克服 GPU 计算吞吐量与调用的瓶颈,大大的增加计算网格的数量,如不考虑时间成本,可以提高到百亿级别;计算时间与计算网格数量的大小呈明显线性关系,而由于多次调用增加的 CPU 与 GPU 通讯时间的开销可以忽略不计;同时可以预见的是,该方法使得计算速度较差的低端图形处理器同样可以克服调用瓶颈,达到扩大计算网格量的目的。

表 2 不同网格数目与线程调用次数情况下的单次迭代时间对比

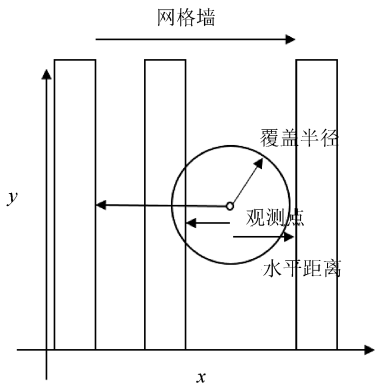
调用次数 \ 网格数目	64000 (40×40×40)	1000000 (100×100×100)	8000000 (200×200×200)	64000000 (400×400×400)	216000000 (600×600×600)
1	0.098 s	1.41 s	单次时间溢出	单次时间溢出	单次时间溢出
5	0.11 s	1.47 s	11.13 s	单次时间溢出	单次时间溢出
20	0.15 s	1.48 s	12.02 s	单次时间溢出	单次时间溢出
50	0.26 s	1.59 s	12.56 s	95.45 s	单次时间溢出
80	0.35 s	1.67 s	12.88 s	96.2 s	319.5 s
100	0.41 s	1.69 s	13.01 s	96.9 s	320.6 s

3.2 敏感范围搜索

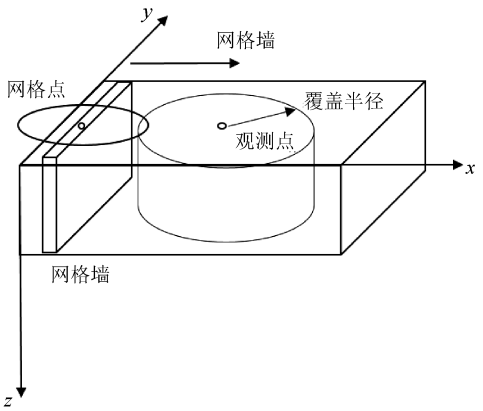
采用的另一种优化技术是敏感度范围搜索,这是一项目前在航空电磁、重力、重力梯度等反演中较为广泛使用的计算优化技术[7,14,15]。不同的地球物理场之间有着不同的影响范围,对于单个观测点而言,在某个范围外的异常体对其造成的影响几乎可以忽略,这个范围就被称作为敏感范围,即 footprint。如重力垂直分量的反演中,可以仅考虑约 20 km 范围以内的密度异常体对观测点的影响,这对于上百公里尺度的航空重力反演而言,可以很大幅度的减少计算量。笔者对于敏感范围搜索的主要创新在于,在多次线程调用代码结构下实现了敏感范围搜索算法。

如图 2 所示,在线程多次调用结构下,每个 GPU 计算节点单次处理的网格数量由整个计算域变成了顺序排列的“网格墙”。我们采用了一种圆

柱形的敏感范围计算方法,即只判定水平距离。当计算 $\boldsymbol{I}_{\text{par}}$ 矢量时,如果观测点的敏感度覆盖半径小于观测点到网格墙水平距离的最小值时,则抛弃整堵“墙”,进入下一次 GPU 调用。反之,则计算每个网格点与观测点的水平距离,判断其是否在计算覆盖半径以内(图2a)。当计算 $\boldsymbol{J}_{\text{par}}$ 矢量时,则计算其对覆盖半径内的所有观测点的影响(图 2b)。



(a) 观测点覆盖半径(footprint)俯视



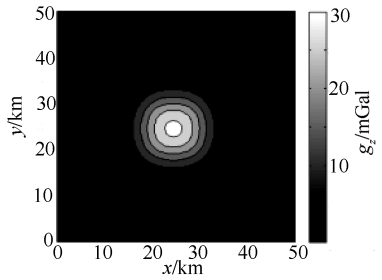
(b) 观测点与网格点覆盖半径(footprint)三维示意

图 2　线程多次调用结构下的 GPU 并行敏感范围搜索示意

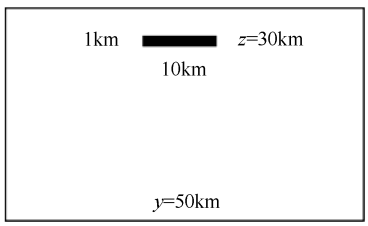
综上,多次调用线程的使用,提高了并行算法的最大适用网格数。在此基础上引入的敏感范围搜索,又进一步在面对大尺度的重力反演中,成比例的减少迭代计算时间。

4　模型计算

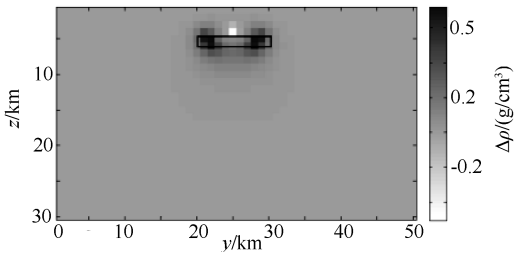
为验证并行程序的正确性与有效性,设计一个长、宽均为 50 km,深度 30 km 的反演区域,区域中心安放了密度异常为 $\Delta\rho = 1\text{g/cm}^3$,长、宽、高分别为10 km、10 km与1km的密度异常体薄板,x 方向中轴



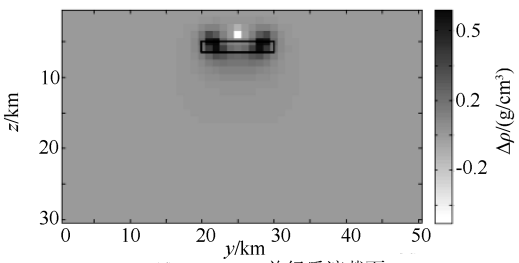
(a) 正演数据等值线



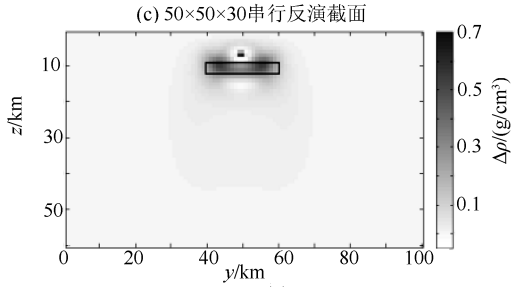
(b) 异常体位置截面



(c) 50×50×30串行反演截面



(d) 50×50×30并行反演截面



(e) 100×100×60并行反演截面

图 3　模型计算参数与结果

截面如图 3b 中所示,正演网格采用了 $x \cdot y \cdot z$ 三方向 100×100×60 设计,边长为 500 m,得到正演拟合等值面如图 3a 中所示。

分别采用了 50×50×30 与 100×100×30 的网格数量进行反演计算,并 50×50×30 传统串行反演进行比对。为验证优化算法正确性,并行代码中强行设置了分五次调用 GPU 线程,并设置了 20 km 的敏感度范围,而串行代码中使用常规 RRCG 方法。反演设定误差小于 5%时停止迭代,结果如图 3 中所示。同等网格条件下的并行计算反演结果与传统 RRCG 方法所得结果基本一致,验证了上述优化算法代码的正确性。而在传统 RRCG 算法已经内存溢出的细网格条件下,并行计算结果更真实的反映了异常体的形态与峰值,体现了在大尺度重力计算中,优化后的并行计算在反演精度上的优势。

5 结论与建议

1) 通过 CUDA 语言 GPU 的并行算法设计,提高了 RRCG 反演计算的效率与计算容量;采用优化算法进一步提高了海量三维网格处理的能力与计算速度。算法通过了模型计算比较,结果正确可信。该方法为解决重力大型三维反演提供了一种思路。

2) 未来工作中可以将该方法推广到实际的大尺度海洋、航空重力反演中;算法方面,可以进一步开发磁法与重磁张量反演,以及重磁联合反演和三维地质建模。

参考文献:

[1] Lü Q,Qi G,Yan J.3D geologic model of Shizishan ore field constrained by gravity and magnetic interactive modeling:A case history[J].Geophysics,2012,78(1):B25-B35.

[2] Wang G,Zhang S,Yan C,et al.Mineral potential targeting and resource assessment based on 3D geological modeling in Luanchuan region,China[J].Computers & Geosciences,2011,37(12):1976-1988.

[3] 姚长利,郝天珧.重磁遗传算法三维反演中高速计算及有效存储方法技术[J].地球物理学报,2003,46(2):252-258.

[4] 陈召曦,孟小红,刘国峰,等.基于 GPU 的任意三维复杂形体重磁异常快速计算[J].物探与化探,2012,36(1):117-121.

[5] Chen Z,Meng X,Guo L,et al.GICUDA:A parallel program for 3D correlation imaging of large scale gravity and gravity gradiometry data on graphics processing units with CUDA[J].Computers & Geosciences,2012,46:119-128.

[6] 林巍,张健,李家彪.南沙中业群礁地区中生代残留盆地 GPU 重力异常三维反演[J].热带海洋学报,2013,32(4):36-42.

[7] Cuma M,Wilson G A,Zhdanov M S.Large-scale 3D inversion of potential field data[J].Geophysical Prospecting,2012,60(6):1186-1199.

[8] Cuma M,Zhdanov M S.Massively parallel regularized 3D inversion of potential fields on CPUs and GPUs[J].Computers & Geosciences,2014,62:80-87.

[9] Zhdanov M S.New advances in regularized inversion of gravity and electromagnetic data[J].Geophysical Prospecting,2009,57(4):463-478.

[10] Portniaguine O,Zhdanov M S.Focusing geophysical inversion images[J].Geophysics,1999,64(3):874-887.

[11] Zhdanov M S.Geophysical inverse theory and regularization problems[M].Elsevier,2002.

[12] Cook S.CUDA programming:a developer's guide to parallel computing with GPUs[M].Newnes,2012.

[13] Hoshino T,Maruyama N,Matsuoka S,et al.CUDA vs openACC:Performance case studies with kernel benchmarks and a memory-bound CFD application[C]//Cluster,Cloud and Grid Computing (CCGrid),2013 13th IEEE/ACM International Symposium on.IEEE,2013:136-143.

[14] Zhdanov M S,Smith R B,Gribenko A,et al. Three-dimensional inversion of large-scale EarthScopemagnetotelluric data based on the integral equation method: Geoelectrical imaging of the Yellowstone conductive mantle plume[J].Geophysical Research Letters,2011,38(8):1-7.

[15] Cox L H,Wilson G A,Zhdanov M S.3D inversion of airborne electromagnetic data using a moving footprint[J].Exploration Geophysics,2010,41(4):250-259.

Regularized inversion of 3D gravity data:a new GPU parallelized method based on CUDA

LI Wu-Yang[1,2],ZHANG Jian[1,2],LIN Wei[3]

(1.*College of Earth Science,University of Chinese Academy of Sciences,Beijing* 100049; 2.*Key Laboratory of Computational Geodynamics,Chinese Academy of Sciences,Beijing* 100049; 3.*Department of Geology and Geophysics,University of Utah,Salt Lake City* 84112)

Abstract:We introduce a new 3D parallelgravity inversion method based on CUDA GPU in PGI Fortran,which could be used in PCs and Laptops with NVDIA graphic cards to accelerate iteration speed for Re-Weight Regularized Conjugate Gradient method up to hundreds times.Storage and threads optimization was made for visual OS, leading a more than 0.1 billion cell's number for inversion in PCs.The result of model study shows that this method is an efficient and believable parallel computing algorithm.

Key words: gravity;3D inversion;GPU;CUDA; parallel computing;RCG method

作者简介: 李午阳(1990-),男,现为中国科学院大学在读博士生,主要研究方向为非震地球物理正反演。