栅格数据处理中邻域型算法的并行优化方法*

程 果,景 宁,陈 荦,熊 伟,欧阳柳
(国防科技大学 电子科学与工程学院,湖南 长沙 410073)

摘 要:随着并行计算的成熟,众多数据密集型的栅格处理算法亟需利用并行计算来缩减执行时间。针对其中一类邻域型算法,构建了用于估计是时间代价的串行/并行时域模型,分析了各个组成的代价影响因素,提出了降低数据I/O代价的并行I/O方法和降低数据通信代价的光圈预测方法。实验证明,所提的两个优化方法可以使邻域型栅格处理算法的并行程序更加充分地利用并行计算资源,进而在一般并行化的基础上进一步提升其并行性能。

关键词:栅格数据处理;邻域型;并行I/O;光圈预测;MPI
中图分类号:TP311 **文献标志码**:A **文章编号**:1011-2486(2012)04-0114-06

Parallel optimization methods for raster data processing algorithms of neighborhood-scope

CHENG Guo, JING Ning, CHEN Luo, XIONG Wei, OUYANG Liu
(College of Electronic Science and Engineering, National University of Defense Technology, Changsha 410073, China)

Abstract: As parallel computing has become mature and practical, data intensive raster data processing algorithms are desiderating parallel computing technologies to reduce the running time. The objectives of this research focuses on the parallelization of neighborhood-scope algorithms. the sequential/parallel temporal model was developed, the affecting factors of each component of the temporal model were analyzed, and two optimization methods were proposed, which can further promote the parallel performance of neighborhood-scope algorithms: the Parallel I/O method that can reduce the data I/O cost; and the Halo Prediction method that can reduce the data communication cost. Experiments verified the effectiveness and efficiency of the proposed optimization methods, which can further promote the parallel performance by making the parallel algorithmic program fully take advantage of parallel computing resources.

Key words: raster data processing; neighborhood-scope; Parallel I/O; Halo prediction; MPI

从1850年Charles Babbage第一次提出并行计算[1],160多年来软、硬件层面的飞速发展,使得并行计算技术已经成为解决计算密集型和数据密集型问题的有效手段。栅格数据处理为一类典型的数据密集型和计算密集型计算,产生了许多利用并行计算技术来缩减栅格数据处理算法运行时间的研究成果[2-3]。参照Tomlin对栅格数据处理算法的分类,数量种类繁多的栅格处理算法根据其算子计算特征可分为局部型、邻域型、区域型和全局型[4]。其中邻域型栅格处理算法是本文的研究对象。

并行地理计算经过15年的发展,其研究路线大致归为两个方向[5]:(1)并行改造已有串行算法,进而实现计算密集型和/或数据密集型地学问题的快速求解;(2)研发适合当前新型硬件架构的新算法,进而实现计算密集型和/或数据密集型地学问题的快速求解。本文的研究将遵循第一个方向,通过对传统串行栅格处理算法进行并行化改造,并从数据I/O和数据通信两方面寻求可提升算法并行性能的优化方法。

当前主流的并行计算模式主要有:(1)基于消息传递模型的进程级并行,如MPI;(2)基于共享内存模型的线程级并行,如OpenMP;(3)CPU、GPU混合并行,如CUDA;(4)还有其他诸如Map-Reduce[6]、网格计算[7]和云计算[8]。其中,MPI不仅可以应用于多样的计算环境,而且包含一整套完善的消息传递机制服务于涉及通信的算法并行化,已成为地学界广为采用的并行化模式,也是本文研究并行优化方法的基本模型。

* **收稿日期**:2011-09-01
基金项目:国家863计划资助项目(2011AA120300);国家自然科学基金资助项目(40801160, 61070035);高等学校博士学科点专项科研基金项目(20104307110017)
作者简介:程果(1984—),男,河北邯郸人,博士研究生,E-mail:guocheng@nudt.edu.cn;
景宁(通信作者),男,教授,博士,博士生导师,E-mail:ningjing@nudt.edu.cn

1　时间代价估计模型

1.1　邻域型栅格处理算子

栅格处理算法大多数都是按照某一给定算子对栅格数据集的每一个点执行遍历式计算。在计算过程中,由于算子涉及的数据集范围不同,可将算法分为局部型、邻域型、区域型和全局型。其中,邻域型算法指的是算子的计算输入数据包括计算点的数据,以及以计算点为中心的某一指定邻域的所有点的数据。图 1 展示了三个常见的邻域型算子。以图 1(b)子图所示的 Morre 型算子为例,对任意点 $p(i,j)$(灰色单元)执行算子计算时,不仅依赖于该点数据 $d(i,j)$,还需要用到以该点为中心的 Morre 邻域内的所有点(条纹单元)的集合。

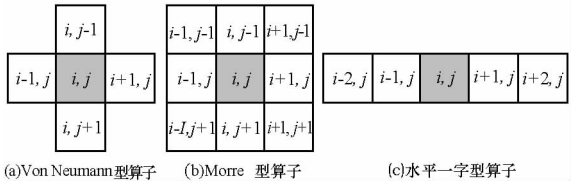

图 1　邻域型算子举例

Fig. 1　Examples of neighborhood-scope operators

1.2　串行时域模型

一个传统的栅格处理算法的串行计算流程中主要包括三个步骤:(1)数据导入,即将数据从磁盘中的栅格文件读至内存;(2)算子计算,即参照算子方程对所有数据点执行计算;(3)结果导出,即将计算后的结果数据从内存写至磁盘中的结果文件。这样一来,串行计算的时间代价应包含上述三个部分,其时域模型为:

$$t_{srl} = t_{load} + t_{cpt} + t_{wrt} \tag{1}$$

其中,t_{read}、t_{cpt} 和 t_{wrt} 分别表示上述的数据导入、数据计算和结果写出时间,N_{ds} 表示输入数据集规模。定义数据导入速率(v_{read})、数据计算速率(v_{cpt})和数据写出速率(v_{wrt})。则式(1)进一步细化成:

$$t_{srl} = N_{ds}(1/v_{read} + 1/v_{cpt} + 1/v_{wrt}) \tag{2}$$

上述串行时域模型清楚地指出,输入数据规模成为影响时间代价的一项重要因素。因此,大规模栅格数据处理算法将是一项耗时的数据密集型计算。面对数据密集型计算,基于数据划分的并行化是一项有效地加速手段。下面提出基于数据划分的并行化方法和相应的并行时域模型。

1.3　并行时域模型

数据划分指的是按照一定的方法对原始数据集进行切割。基于数据划分的并行计算指的是通过将规模较大的原始数据集划分生成多个小规模子数据集,进而分配给多个计算单元实现并行计算。计算时间取决于所有并行计算的最长时间。参考串行时域模型,并行时域模型应为:

$$\begin{aligned} t_{para} &= t_{pload} + t_{cpt} + t_{pwrt} \\ &= t_{pload} + \max_{i\in[1,n_p]}(N_{sds,i}/v_{cpt,i}) + t_{pwrt} \end{aligned} \tag{3}$$

其中 n_p 表示并行计算单元数(在不同的计算环境中,可以代表为节点数、处理器数和计算核数,为了便于表述,本文之后统一采用核数这一说法);$v_{cpt,i}$ 表示第 i 号核的计算速率;表示分配给的第 i 号核的子数据集规模;t_{pload} 表示数据从磁盘加载到所有每一个核的内存中所用的时间,即数据并行导入时间;而 t_{pwrt} 表示结果从所有每一个核的内存中写至磁盘所用的时间,即结果并行写出时间。

邻域型算法在基于数据并行思想的并行化改造后,其并行计算阶段将会引入通信操作。如图 2 所示,对于一个邻域型算子,当执行十字型数据划分方法后,原始数据集被划分成 4 个子数据集,并分配至 $P1 \sim P4$ 四个核执行并行计算。由于 $P1 \sim P4$ 每个核都只获取到了负责计算的子数据集,而对于其余子数据集内的数据无权访问,光圈效应[9]由此而生。

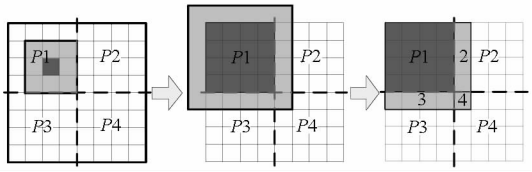

图 2　光圈效应带来数据通信

Fig. 2　Halo phenomenon brings data communications

定义 1　光圈效应(Halo Phenomenon)

当多个核在对数据划分后的子数据集执行邻域型算子的并行计算时,其中的任一核的计算不仅需要该核已分配拥有的子数据集,还需要一部分其他核所分配拥有的数据点。这些额外的数据点通常会组成一个圆环形状的光圈,因此称之为光圈效应。

光圈效应将会在并行计算中引入通信代价。相应的并行时域模型修改为:

$$\begin{aligned} t_{para} &= t_{pload} + \max_{i\in[1,n_p]}(N_{sds,i}/v_{cpt,i}) \\ &\quad + \max_{i\in[1,n_p]}(t_{cmm,i}) + t_{pwrt} \end{aligned} \tag{4}$$

其中相较于式(3)增加的变量 $t_{cmm,i}$ 表示第 i 号核和其他核的通信总时间。

并行时域模型的提出,使得并行栅格数据处理算法的时间代价组成清晰可见。下文将提出两个

优化方法用于降低其中的 I/O 代价和通信代价。

2 并行 I/O 方法

数据导入时间 t_{pload} 指的是所有核并行地完成数据从磁盘加载到内存所用的时间,它的值取决于并行数据导入的方法。常见的传统方法之一是基于主从式结构的数据分发方法,即由主核(控制核)将数据从磁盘中的文件读取至主核的内存,而后按照某一划分方法(行划分、列划分或十字划分等)将原始数据集划分为若干子数据集,之后采用通信的方式将子数据集发送至相应的从核(计算核)的内存,完成数据并行导入。采用数据分发方法下的数据导入时间的计算式应为:

$$t_{pload}^{dis} = t_{read} + t_{part} + t_{dis} = N_{ds}/v_{read} + t_{part} + t_{dis} \quad (5)$$

其中 t_{part} 表示数据划分时间,通常取决于划分方法,一旦数据规模和划分任务确定,其值基本保持不变;t_{read} 表示控制核读取数据至内存的时间,其值为数据集规模(N_{ds})和数据读取速率(v_{read})的比值;t_{dis} 表示控制核向计算核分发子数据集的时间。采用数据分发方法完成数据导入时,控制核向多个计算核发送子数据集是时间代价很大的瓶颈问题。为此,下文提出了并行 I/O 方法,可以有效地消除上述通信瓶颈。

并行 I/O 方法的提出受到了 Google File System(GFS)的启发。GFS 一文中提到[10],为了避免数据读写成为性能瓶颈,用户(client)不再直接从/向主节点(master)读/写数据,取而代之的是,用户向主节点询问和其相关联的数据服务器节点(chuckserver),而后在控制信息(control message)的帮助下,直接和数据服务器完成数据读写。这样一来,用户和主节点之间避免了多用户并发冲突,大幅度提高了数据读写效率。

数据并行导入在逻辑层面上有很多和 GFS 相似的元素。在主从式结构中,存在一个控制核和多个计算核。控制核的角色类似于 GFS 中的主节点,而计算核相当于 GFS 中的用户。数据的并行导入要求每个计算核通过控制核获取分配给自己的子数据集,这就相当于 GFS 中用户向主节点请求数据。那么多个计算核和控制核之间的并发通信则类似于 GFS 中的多用户并发数据读写一样,成为影响性能的瓶颈问题。下面参考图 3,详述借鉴于 GFS 思想的并行 I/O 方法。

并行 I/O 方法采用共享文件系统作为底层数据存储系统,存储在共享文件系统中的数据文件可以由该计算环境内的任何核访问。并行吞吐性能较好地共享文件系统,例如当前高性能集群中

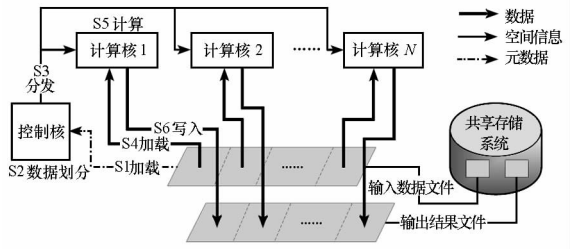

图 3 并行 I/O 方法流程

Fig. 3 The workflow of the parallel I/O method

常见的 General Parallel File System (GPFS)[11],将会提升并行 I/O 方法的实际效果。并行 I/O 方法也是基于主从式结构的。方法的第一步是由控制核读取输入数据文件的元数据,从而获取输入 DEM 数据的空间范围信息;而后依然是控制核基于空间范围执行数据划分,不过这里的划分不是传统地切割原数据集生成子数据集,而是通过空间范围划分,生成表示子数据集空间范围的子元数据(sub-metadata);第三步是将数据划分生成的子元数据分发至各个计算核;而后在子元数据提供的空间信息帮助下,计算核直接访问共享存储系统中输入数据文件的对应空间范围内的数据,完成数据并行导入。采用并行 I/O 方法实现数据导入的时间代价计算公式应为:

$$\begin{aligned} t_{pload}^{pio} &= t_{m-read} + t_{part} + t_{m-dis} + t_{pread} \\ &= t_{m-read} + t_{part} + t_{m-dis} + N_{sds}/v_{pread} \end{aligned} \quad (6)$$

其中,t_{part} 依然表示数据划分时间;t_{m-read} 表示控制核的读取元数据的时间;t_{dis} 表示控制核向计算核分发子元数据的时间;t_{p-read} 表示计算核从共享存储系统并行读取子数据集的时间,其值应为子数据集规模(N_{sds})和数据并行读取速率(v_{pread})的比值。下文将证明采用并行 I/O 方法实现数据并行导入的时间代价小于传统的数据分发方法的时间代价。

由经典模型 LogP[12] 和 LogGP[13] 可知,

$$\begin{aligned} t_{cmm} &= t_{init-send} + t_{pass} + t_{init-recv} \\ &= t_{init-send} + L_{msg}/v_{cmm} + t_{init-recv} \end{aligned} \quad (7)$$

即一次消息通信时间(t_{cmm})等于发送初始化时间($t_{init-send}$)、接受初始化时间($t_{init-recv}$)和消息传递时间(t_{pass})之和。其中消息传递时间取决于消息长度(L_{msg})和两个核之间的传输速率(v_{cmm})。在数据分发方法中,控制核向计算核发送子数据集的数据,所以子数据集规模(S_{sd})即为消息长度,即 $L_{msg} = S_{sd}$;而在并行 I/O 方法中,控制核向计算核发送子元数据,那么子元数据规模(S_{msd})即为消息长度,即 $L_{msg} = S_{msd}$。因此可得:

$$\begin{cases} t_{dis} = t_{init-send} + S_{sd}/v_{cmm} + t_{init-recv} \\ t_{m-dis} = t_{init-send} + S_{msd}/v_{cmm} + t_{init-recv} \end{cases} \quad (8)$$

为了证明并行 I/O 方法的时间代价更小,只需证明:式(6) -式(5) <0,而

$$\begin{aligned} t_{pload}^{pio} - t_{pload}^{dis} &= (t_{m-read} + t_{part} + t_{m-dis} + N_{sds}/v_{pread}) \\ &\quad - (N_{ds}/v_{read} + t_{part} + t_{dis}) \\ &\approx (S_{msd} - S_{sd})/v_{cmm} + N_{sds}/v_{pread} \\ &\quad - n_p N_{sds}/v_{read} \end{aligned} \tag{9}$$

定义并行存储系统的并行读取冲突指数 $k_{pread} = v_{read}/v_{pread}$。当 k_{pread} 为最小值 1 时是理想状态,表示无并行读取冲突,数据的多核并行读取速率等于单核串行读取速率。k_{pread} 越大,表示冲突越严重,意味着该共享存储系统的并行吞吐性较差。对于一般共享存储系统,$1 < k_{pread} < n_p$。将其代入式(9),得:

$$\begin{aligned} &t_{pload}^{pio} - t_{pload}^{dis} \\ &= (S_{msd} - S_{sd})/v_{cmm} + (k_{pread} - n_p)N_{sds}/v_{read} < 0 \end{aligned} \tag{10}$$

综上可见,并行 I/O 方法相比于数据分发方法,数据并行导入代价更小。

数据并行导入的成功优化启发我们进一步把目光转向结果并行导出部分。在并行 I/O 方法中,计算核在完成数据计算后,不再需要将子结果集发回给控制核完成汇总,而是利用前面接收过的子元数据,直接根据空间范围写入至共享存储系统中结果文件的相应部分。这样一来,子结果集通过通信完成汇总的时间代价完全被节省下来。而且相比于原先的控制核统一写结果的模式,子结果集由多个计算核并行写出,时间代价也会有一定的缩减。

3　光圈预测方法

本节提出的方法旨在优化通信代价。对于邻域型算法,基于数据划分的并行化后,需要在计算过程中利用通信来传递光圈数据。也就是说,在好的方面,多核并行计算降低了计算时间代价;而在坏的方面,额外引入的通信代价不可忽视。尤其是当面对一个低速通信网络环境时,引入的通信代价甚至可能会超过降低的计算代价,从而严重影响算法的并行性能指标。

通信的源头是光圈效应。只要能设法消除光圈效应,即可降低上述核心时间代价。通过进一步分析光圈效应,我们总结出了光圈的形状是由数据划分方法,划分任务数目,以及邻域形状决定的。只要确定上述信息,光圈的形状、大小和区域即可被提前预测出;而后,在数据划分过程中,将预测出的光圈数据补充到相应的子数据集,即增大子元数据包含的空间范围;随后,通过运用并行 I/O 方法,各个计算核参照元数据信息,在读取子数据集的同时,一并读取光圈数据,从而避免了传递光圈数据的通信操作。这样一来,通信操作被替换成了代价较小的光圈预测和光圈预取操作,程序运行时间又一次被缩减。由图 4 可见,一个完整的光圈包括两个横边、两个纵边和四个角,对于给定的一个邻域算子,光圈计算方程为:

$$\begin{aligned} N_{halo} &= 2 \cdot N_{edge-x} + 2 \cdot N_{edge-y} + 4 \cdot N_{corner} \\ &= 2wr_y + 2hr_x + 4r_x r_y \end{aligned} \tag{11}$$

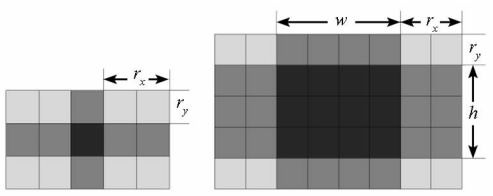

图 4　光圈预测图示

Fig. 4　The illustration of the halo prediction

然而在实际的数据划分过程中,不同的划分方法和划分数目可能会生成不同形状和组成的光圈。如图 5 所示,本文对经典的行划分、列划分和十字划分在不同划分数目下可能产生的光圈进行了分析并总结归纳出一共 8 种不同类型形状和组成的光圈。

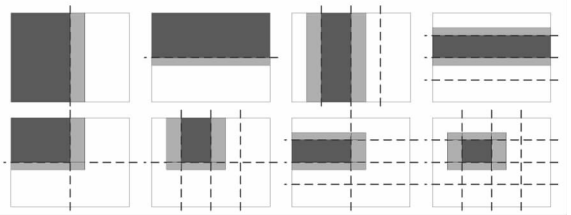

图 5　不同划分策略产生的不同形状和组成的光圈

Fig. 5　Different data-partitioning methods produce halos of different shapes and sizes

综上可见,对于邻域型算法,光圈预测方法可以将光圈通信操作优化为耗时更短的光圈预取操作。联合使用所提出的并行 I/O 方法,光圈预取时采用并行读取方法,即可保证光圈预取的时间代价远远小于采用光圈通信的时间代价。但是这里需要补充的是,该方法只在针对单一算子的简单算法时最为有效。对于包含多个算子的复杂算法,光圈预测只能有效地降低算法流程中第一个算子的时间代价;而对于靠后的算子,由于预测复杂造成的预测代价过高,或是预测的光圈范围过大造成的并行读取代价过高,使得该方法的效果降低。Guan[14] 提出了适用于多算子复杂算法的 edge first 方法,我们会在下一步研究中借鉴其经验,研究多算子复杂栅格处理算法的并行优化。

4 实验分析

4.1 实验环境

为了验证本文提出的两个并行优化方法的有效性和高效性,我们基于 IBM 高性能计算集群实施了对比实验。集群包括 1 个主控节点(x3650M3),4 个 I/O 节点(x3550M3),28 个计算节点(Blade HS22)以及 Infiniband 网络和 GPFS 文件系统。实验数据采用了 2006 年美国大陆地区高程数据,尺寸为 21 000 × 13 000,空间分辨率为 250m。我们选择了栅格数据处理中地形分析领域的坡度坡向计算作为实验用例。

4.2 串行/并行对比实验

首先实施了坡度坡向串行算法[15]和不采用任何优化方法并行算法的性能对比实验。以下所有的实验结果图中,核数为 1 的结果值表示串行算法的测试结果,其余是并行算法在不同核数下的测试结果。

图 6 展示了串行和不同核数下并行的性能对比。并行后的执行时间随着处理器数目的增加而减少,其降低趋势会趋于平缓。这是由于核数的增加只会减少执行时间中的数据计算时间,而对数据导入、结果导出时间的影响几乎可以忽略。当核数足够大后,数据计算时间已经足够小,使得数据导入、导出时间成为主影响因素,总时间不再减少。同理,并行加速比增长也趋于平缓。

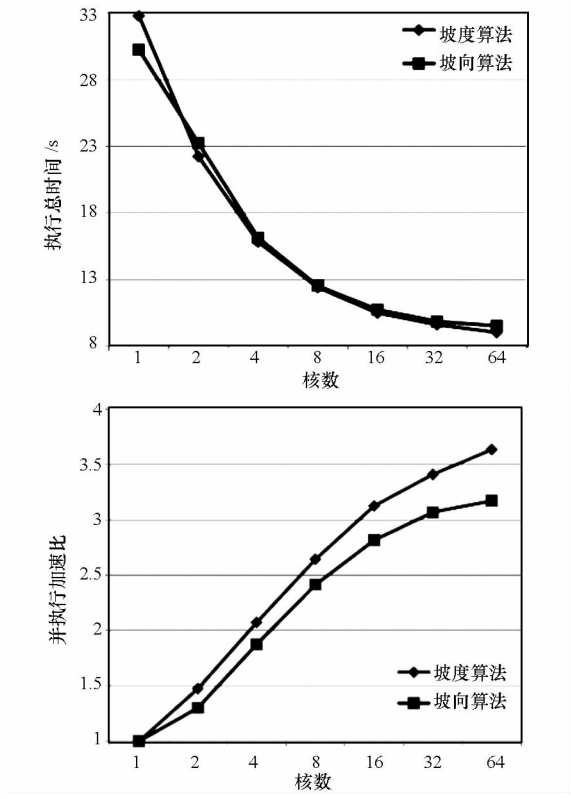

图 6 坡度坡向串行算法和并行算法的性能对比

Fig. 6 The performance comparison between the sequential and parallel slope and aspect algorithms

4.3 优化/非优化对比实验

为了证明本文提出的并行 I/O 方法和光圈预测方法对邻域型算法的并行优化效果,在上节的实验基础上,我们进一步对坡度坡向并行算法进行优化,设计实现了使用并行 I/O 方法的单优化算法和联合采用并行 I/O 及光圈预测方法的双优化算法,并对比这三种算法的并行性能。图 7 和图8的实验结果证明:在本实验环境下单优化算法性能提升显著,即并行 I/O 方法的贡献突出,可以大幅度降低执行时间,从而提高并行加速比和并行效率。而双优化算法相比于单元化算法只有略微提升,这是因为本实验环境中的通信网络是高速 Infiniband,高通信速率使得通信时间很短,光圈预测方法节省的通信代价不够显著。如

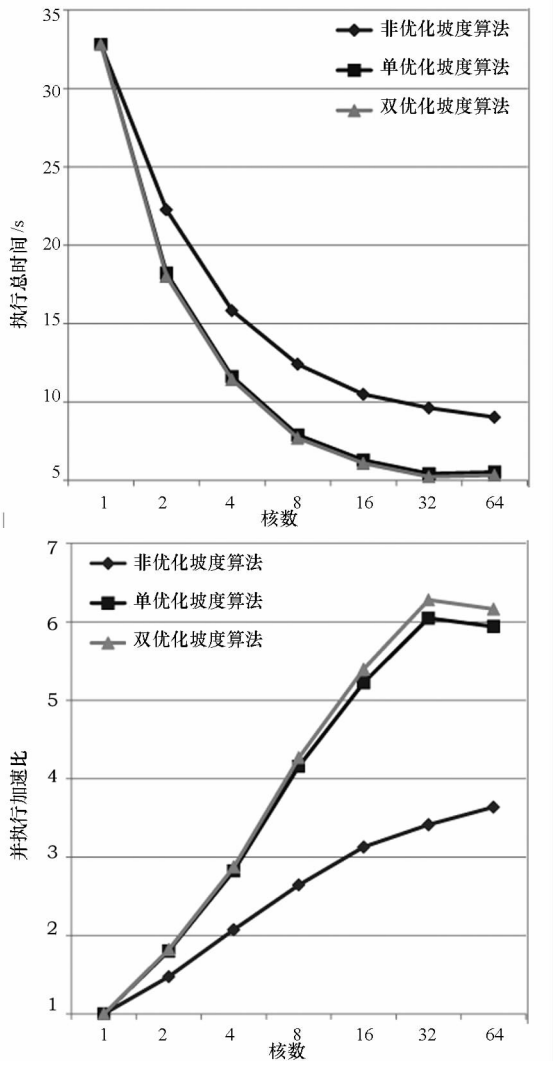

图 7 是否使用优化的坡度算法并行性能对比

Fig. 7 The performance comparison between the slope algorithms using and non-using optimization methods

果实验环境是低速网络,相信光圈预测方法的贡献也会比较显著。

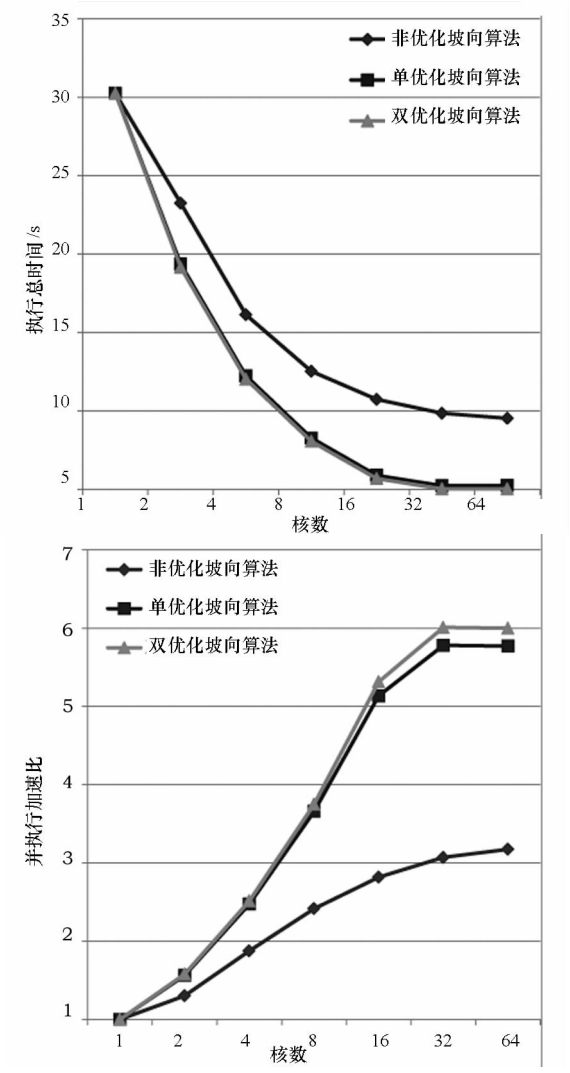

图 8　是否使用优化的坡向算法并行性能对比

Fig. 8　The performance comparison between the aspect algorithms using and non-using optimization methods

总之,本文所提的两个优化方法可以提升邻域型算法的并行性能。当超过 16 个核后,加速比和效率都提高了一倍左右。

5　小　结

本文基于消息传递模型,选择 MPI 作为并行模式,采用数据并行思想,研究了邻域型栅格处理算法的并行化技术。并行化容易、并行优化难。一个好的并行算法应该能够最大限度地利用并行计算资源来提高算法效率。本文在构建串行和并行时域模型的基础上,分析了各个时间代价组成部分的影响因素,研究并提出了可降低数据 I/O 代价的并行 I/O 方法和可降低数据通信代价的光圈预测方法。实验证明,在一般并行化的基础上,本文提出的优化方法可以进一步提升邻域型栅格处理算法的并行性能。本文研究成果可以为广大地学研究者研发并行栅格处理算法提供有益的建议和指导。

参考文献(References)

[1] Kuck D J. High performance computing: challenges for future systems[M]. New York, NY: Oxford University Press, 1996.

[2] Guan X F, Wu H Y. Leveraging the power of multi-core platforms for large-scale geospatial data[J]. Computers & Geosciences, 2010, 36:1276 – 1282.

[3] Huang Q Y, Yang C W. Optimizing grid computing configuration and scheduling for geospatial analysis: an example with interpolating DEM [J]. Computers & Geosciences, 2011, 37(2):165 – 176.

[4] Tomlin C D. Geographic information systems and cartographic modelling[M]. Englewood Cliffs, NJ: Prentice Hall, 1990.

[5] Clematis A, Mineter M, Marciano R. High performance computing with geographical data[J]. Parallel Computing, 2003, 29(10):1275 – 1279.

[6] Jeffrey D, Sanjay G. MapReduce: simplified data processing on large clusters[C]//Proceedings of OSDI′04. USENIX Association, 2004:137 – 150.

[7] Foster I, Kesselman C, Tuecke S. The anatomy of the grid: enabling scalable virtual organizations[J]. International Journal Supercomputer Applications, 2001, 15(3):200 – 222.

[8] Armbrust M, Fox A, Griffith R, et al. Above the clouds: a berkeley view of cloud computing[R]. Berkeley, 2009.

[9] Mineter M J. Partitioning raster data[M]. Bristol, PA: Taylor & Francis, 1998.

[10] Ghemawat S, Gobioff H, Leung S T. The google file system [C]//Proceedings of the 19th ACM symposium on Operating Systems Principles,New York, NY, USA: ACM, 2003:29 – 43.

[11] Schmuck F, Haskin R. GPFS: a shared-disk file system for large computing clusters[C]//Proceedings of the Conference on File and Storage Technologies (FAST), USENIX Association, 2002:231 – 244.

[12] Culler D, Karp R, Patterson D, et al. Logp towards a realistic model of parallel computation[C]//Proceedings of the 4th ACM SIGPLAN Symposium on Princi-Plesand Practice of Parallel Programming,New York, NY, USA: ACM Press, 1993:1 – 12.

[13] Alexandrov A, Ionescu M F, Schauser K E, et al. LogGP: incorporating long messages into the LogP model for parallel computation [J]. Journal of Parallel and Distributed Computing, 1997, 44(1):71 – 79.

[14] Guan Q F. pRPL an open-source general-purpose parallel raster processing programming library[J]. Sigspatial Special, 2009, 1(1):57 – 62.

[15] 刘学军,龚健雅,周启鸣,等. 基于 DEM 坡度坡向算法精度的分析研究[J]. 测绘学报,2004,33(3):258 – 263.
LIU Xuejun, GONG Jianya, ZHOU Qiming, et al. A study of accuracy and algorithms for calculating slope and aspect based on grid digital elevation model(DEM)[J]. Acta Geodaetica et Cartographica Sinica, 2004, 33(3):258 – 263.