一种面向大型地理栅格数据的并行处理框架*

杨典华[1],潘 欣[2]
(1. 首都师范大学 三维信息获取与应用教育部重点实验室, 北京 100048;
2. 中国科学院 东北地理与农业生态研究所,吉林 长春 130102)

摘 要:随着高分辨率数据获取技术的发展,地理栅格数据的数据量不断增大,串行计算方式无法快速处理大型栅格数据,因此需要通过并行技术提高效率。传统开发过程将算法与进程调度、内存管理以及数据I/O混杂在一起的编程方式,对程序员要求较高,代码质量不易控制。提出了一种面向大型地理栅格数据的并行处理框架,利用核心类的真实和虚拟两种读取方式,实现了大型数据的分步骤、分块的快速加载和写入,并将所有的并行任务调度、进程间的数据传输过程以及特定的栅格算法步骤归结为任务;通过该框架可以将算法本身与并行调度、磁盘I/O等底层操作分离,使算法编写者可以专注于算法本身,降低开发难度,提高代码质量,解决了快速编写大型地理栅格数据算法程序的目的。实验表明,本框架可实现较好的并行效果,并显著降低代码量、提高软件质量。

关键词:栅格图像;超大型地理数据;MPI;并行处理框架
中图分类号:TP391 **文献标志码**:A **文章编号**:1001-2486(2013)06-0152-05

Parallel processing framework for huge geographic raster data

YANG Dianhua[1], *PAN Xin*[2]
(1. Key Laboratory of 3D Information Acquisition and Application, MOE, Capital Normal University, Beijing 100048, China;
2. Northeast Institute of Geography and Agroecology, Chinese Academy of Sciences, Changchun 130102, China)

Abstract: With the advance of technology, geographic raster data's amount increases continuouslly. Single process cannot process large raster data efficiency, so it is necessary to adopt parallel processing. Traditional development method mixes algorithm, processes scheduling, memory management and data I/O together, thus it presents higher requirements for programmers and the code quality is difficult to control. This study proposes a Huge Geographic Raster Data Parallel Processing Framework (HGRDPPF). With the use of core class's real read and virtual read method, framework can achieve a large raster data's fast loading and writing by steps or blocks, and can achieve parallel task scheduling, data transfer and specific algorithm stage into tasks; through this framework, the raster file is split into sub-tasks according to the ability of computer in the cluster, and separate the raster processing algorithm from MPI API, disk IO and logic, developers can concentrate onto the algorithm itself, and achieve higher program quality. Experiments show that this framework can significantly reduce the amount of code while improving software quality, and to achieve a better parallel performance.

Key words: raster image; huge geographic data; MPI; parallel processing framework

随着高分辨率数据获取技术的进步,地理数据的分辨率不断提高,为我们在较小尺度上观察和分析地表的细节信息带来帮助[1]。较高的分辨率、较短的间隔时间、多样的数据来源为我们提供丰富的数据信息的同时,也带来了海量的数据处理任务。面对这些任务,传统在一台计算机上顺序的、单任务的计算方式难以在短时间内获得计算结果[2]。这不但直接影响结果的时效性和用户的体验,也对新的高分辨率技术走向实际应用形成了阻碍。所以有必要改进现有的计算结构,提高运算能力,实现海量栅格数据的快速有效处理。

高性能计算(High-Performance Computing, HPC)通过多CPU、多核心、多计算机集群将一个大型计算任务划分为多个子任务进行分布式并行处理,这已经在很多科学研究领域取得成功[3]。典型的map-reduce计算框架就可以通过统一的分解-合并步骤快速部署算法到并行计算系统[3-6];众多学者通过调用MPI(Massage Passing Interface)接口实现高性能计算,可以显著提高遥感影像处理、遥感影像配准及遥感目标识别等地理算法运行速度[7-11]。虽然以上研究取得了一

* **收稿日期**:2013-07-05
基金项目:国家自然科学基金资助项目(401101384);国家863计划项目(2011AA120302)
作者简介:杨典华(1977—),男,安徽枞阳人,博士研究生,E-mail:yangdh@lreis.ac.cn;
潘欣(通信作者),男,副研究员,博士,E-mail:panxin@neigae.ac.cn

定成果,但是面对大型栅格数据处理还要面临 2 个问题:1)数据处理机制问题,与传统科研计算的“计算密集型”不同,大型栅格数据处理属于“数据密集型”[12],在数据加载上,以往方法均倾向于将数据一次加载于各进程之中,当大型数据本身比内存(包括各个节点内存总和)大时,读取和写入将十分困难;2)软件工程质量问题,已有算法将 MPI 通讯、磁盘 IO 和地理处理算法本身掺杂在一起,形成了各个模块之间的“紧耦合”,代码量大、可读性差,微小变化将引起代码整体的修改,使得相关软件的质量下降[13]。

针对以上问题,本研究提出了一种面向大型地理栅格数据的并行处理框架(Huge Geographic Raster Data Parallel Processing Framework, HGRDPPF),基于C ++ 的面向对象技术构造了一个并行环境下、栅格存取、任务调度与划分的程序框架结构,通过该框架:一方面,可以依据集群中各节点计算能力将栅格数据算法处理的过程划分为多个子任务,实现超大型数据的快速读写;另一方面,将栅格处理算法与并行调度、磁盘 IO 分离,使算法编写者可以专注于算法本身,实现模块间“高内聚低耦合”。

1　栅格数据的表示与访问

面对超大型的栅格数据,传统的一次加载所有数据方式显然不能满足要求。研究期望通过 HGRDPPF 框架达到快速、分块访问栅格数据的目标,并屏蔽在分块访问过程中出现的多种逻辑控制与 API 调用的复杂性。框架采用 PBlock 和 PBlockList 这两个类来实现该目标。

PBlock 类表示某栅格数据的某一段连续位置。如图 1 所示:

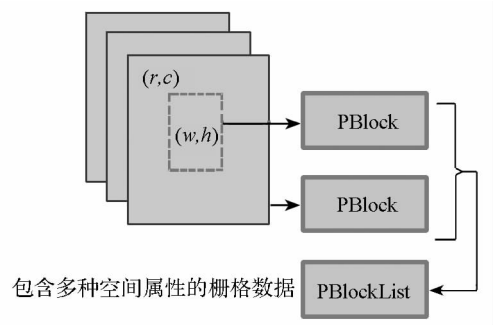

图 1　栅格数据在内存中的表示

Fig. 1　The expression of raster data in memory

每一个 PBlock 对应栅格数据 data(1,2,3,…,n)中空间属性为 i,空间位置为 r 行 c 列的一段连续数据,其宽度为 w 高度为 h。PBlockList 为多个 PBlock 构成的列表,可以实现多个空间属性的共同存取。PBlock 通过封装软件 GDAL 的代码来访问磁盘上的空间栅格数据。PBlock 类的功能和结构主要包含以下几个方面(图 2):

(1)真实和虚拟读取,传统方法通常是直接将所有数据读取到内存之中,与之不同的是 PBlock 支持两种读取方式:虚拟读取 VirtualRead,仅记录它在栅格数据中的位置和大小;真实读取 RealRead,真正将数据读取到内存之中。

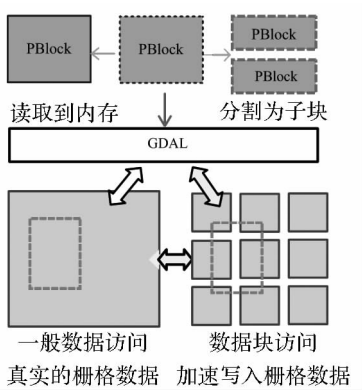

图 2　PBlock 类的功能与结构

Fig. 2　The structure and function of PBlock class

(2)分割与合并:分割方法(split)将 PBlock 再分割为多个子块(如将原有的数据按照行/列分割为 m 块);合并方法(merge)将多个子块合成一个大块。

(3)一般数据访问和分块访问:PBlock 类封装 GDAL,可以直接在相应位置进行读取和写入操作;在(1)、(2)与(3)的支持下,PBlock 可以将一个大型数据分成多个子块,对数据的读写均可以转换到操作子块文件块之上。

其中文件分割算法如下:

SplitRasterFile

输入:大型栅格数据文件 rasterfile

输出:一组分块的文件　splitfiles(1,2,…,n)

begin

1. 建立 PBlock 对象并虚拟读取 rasterfile 全部内容

   Pblock. VirtualRead(rasterfile)

2. 将 block 对象分割为多个子对象

   splitblock(1,2,…,n) = block. split( ) 分割方式可以按行或列分割

3. 对于每个子对象分别进行读取,写入到对应的分块文件

for i = 1. . n

　　splitblock(i). RealRead( );

　　splitblock(i). WriteToSingleBlock(splitfile(i))

　　end

end

将一个大型栅格数据划分为一组子文件之后可以进行分块访问,读取算法如下:

PartlyRead

输入:一组分块的文件 splitfiles(1,2,…,n),带位置大小信息的 PBlock

输出:将数据读取到 PBlock 对应内存之中

begin

1. 找到 PBlock 和分块文件的相交位置,将某个分块文件中相交部分位置信息提取出来

splitblocks(1,2,…,m) = intersect(PBlock, splitfiles(1,2,…,n))

2. 每个 splitblocks 进行真实读取操作读取对应数据

for i = 1…m

splitblock(i). RealRead()

end

3. 将 splitblocks 进行合并

PBlock. Merge(splitblocks(1,2,…,m))

end

对这组文件的写入操作如下:

PartlyWrite

输入:一组分块的文件 splitfiles(1,2,…,n), PBlock 包含数据

输出:将数据写入到分块文件之中

begin

1. 找到 PBlock 和分块文件的相交位置,将某个分块文件中相交部分位置信息提取出来

splitblocks(1,2,…,m) = intersect(PBlock, splitfiles(1,2,…,n))

2. 将每个 PBlock 安装对应位置分割到 splitblocks 之中

PBlock. split(splitblocks(1,2,…,m))

3. 每个 splitblocks 进行写入操作

for i = 1…m

splitblock. Write()

end

end

通过分块文件,可以大大加速超大型文件访问的速度,规避了对整个栅格文件的扫描。在所有栅格算法运行完之后,可以将分块文件进行合并:

MergeRasterFile

输入:一组分块的文件 splitfiles(1,2,…,n)

输出:大型栅格数据文件 rasterfile

begin

1. 建立 PBlock 对象并虚拟读取 rasterfile 全部内容

Pblock. VirtualRead(rasterfile)

2. 对每个子分块对象分别进行读取,写入到 rasterfile 的对应位置

for i = 1…n

splitblock(i). RealRead();

splitblock(i). GetPosition(PBlock);获取其对应位置

end

end

通过 PBlock 可以快速、分块访问栅格数据的目标,并屏蔽在分块访问过程中出现的多种逻辑控制与 API 调用的复杂性。

2 任务划分和总体处理流程

一个栅格处理算法分为 n 个步骤,每个步骤均需要输入栅格数据计算并产生结果。对于栅格数据,无论是读取、写入、传输、计算、块划分都可以看作是输入一个栅格数据并获取对应结果的算法,本研究定义了以下结构:

在 PBlock 和 PBlockList 类基础上,框架包含:

(1)任务 Task 类:所有任务的父类,包含 Run 方法来实现如图 3 所示的过程;

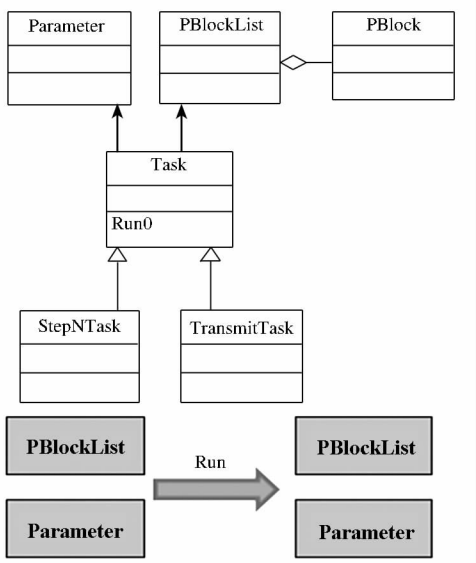

图 3 类图与结构

Fig. 3 Class diagram and structure

Run 方法输入一个 PBlockList 和一个参数管理类 Parameter,经过计算产生结果放入到 PBlockList 类和 Parameter 之中。

(2)算法对应步骤 StepNTask 类:一个栅格处理算法包含 N 个处理步骤,每个步骤可以归结为图 3 所示的过程,所以它们均继承于 Task 类并重写 Run()方法。

(3)传输与同步类 TransmitTask 类:封装 MPI 的数据传输、同步、进程属性等功能,实现 PBlockList 类和 Parameter 类在各个进程之间的传输。

(4)参数处理 Parameter 类:存储并行程序和栅格算法在运行过程中产生的参数。在该模式的

支持下,原有的混杂 MPI、GDAL、C++ 控制代码的栅格处理算法程序将转变为如下过程框架:

MainProcess
输入:大型栅格数据 bigrasterfile
输出:大型栅格数据结果 resultrasterfile
begin
1. 将栅格处理算法,构造一个 Task 子类 TaskNStep 并重写 Run 方法
2. 根据 **SplitRasterFiles** 算法将 bigrasterfile 分割为子块文件
3. 对 PBlock 虚拟读取整个栅格文件
4. 按照当前各计算进程的任务处理能力将 PBlock 进行分割,对于每个分割块进行处理
splitblockTask(1,2,…,m) = BPlock. Split( )
for i = 1. . m
　　splitblockTask(i)调用 **PartlyRead** 算法读取数据
splitblockTaskProcess(1,2,…,k) = splitblockTask(i)再根据进程数进行分割
　　调用 TransmitTask 将 splitblockTaskProcess 同步到各个进程
　　各个进程并行运行 TaskNStep 计算算法并产生结果
　　调用 TransmitTask 将 splitblockTaskProcess 同步进程 0(Rank0)
　　使用 Merge 将 splitblockTaskProcess 合并为 splitblockTask(i)
　　调用 **PartlyWrite** 将 splitblockTask(i)写入到分块文件之中
end
5. 调用 **MergeRasterFile** 算法将分块文件合并为 resultrasterfile 文件并输出结果
end

通过该过程,框架的使用者仅需要关心如何实现栅格算法,将任务划分为多少块,而不用关注如何进行进程间通讯、栅格数据访问 API、逻辑控制等复杂内容,加快软件研发速度,提高软件质量。

3　实验

本框架程序基于 64 位 Linux 环境使用C++实现,MPI 并行环境采用 OpenMPI 1.6,地理数据访问采用 GDAL 1.8。

为了验证框架的有效性,本研究采用了栅格影像的傅里叶变换作为测试算法,傅里叶变换的一个典型特点就是计算每一个像元的结果均需要计算整个影像的所有数据,这就使得单纯“分行加载”方法不能胜任;尤其是面对超大规模栅格数据,由于进程无法一次加载整个影像的数据,所以单纯全部加载方法也不能运行。特殊加载和过程控制使得并行化傅里叶程序复杂度增大,难于实现。

3.1　程序实现和代码对比

傅里叶变换可以分为 5 个步骤:①傅里叶行变换;②傅里叶列变换;③滤波;④傅里叶逆行变换;⑤傅里叶逆列变换。在框架基础上,根据这 5 个步骤分别继承 Task 类建立 5 个步骤子类重写 Run 方法,并依据 MainProcess 提出的处理框架构造一个主函数调用这些类。作为对比,本研究采用一个不使用 HGRDPPF 的程序,在代码量和研发难度上对比如表 1。

表 1　研发难度对比

Tab. 1　The comparison of program difficulty

	使用 HGRDPPF 框架	不使用框架
代码量(精确到百条)	400(不包括框架本身代码)	1300
代码完成后出错并重新调试次数	6	21
研发耗时(小时)	2	6

从表 1 可以看出,通过 HGRDPPF 框架可以以较少代码量和研发周期获得较高质量的代码。

3.2　并行对运行速度的提高

采用测试的硬件环境为 1 台 Intel i5 2300 四核心计算机。通过程序合并了一个 20000 × 20000 栅格作为测试数据,当按照每个栅格用 double 型加载至内存,需要存储空间为:

$$20000 \times 20000 \times 8 = 3051\text{M}$$

对于本文提出的框架,设定单个进程最大占用内存量大小为 200M(显然无法一次装载整个影像),1 -4 进程处理速度如表 2。

表 2　并行运行时间与效率对比

Tab. 2　The comparsion of runing speed and efficiency

进程数	运行时间/s	加速比	并行效率
1	457	1	1
2	252	1.81	0.901
3	175	2.61	0.87
4	141	3.24	0.81

从表 2 可以看出随着进程数的增加,算法运行时间显著降低,在 2 进程时运行时间为 252s 速度接近 1 进程的 2 倍,在 4 进程时速度达到最快 141s。说明通过本框架确实可以利用多进程并行提高运行速度。

但是在并行效率方面,效率随着进程数的增加而下降。在 2 进程时并行效率为 0.901 接近于 1,而在 4 进程时并行效率仅为 0.81。这种效率

的下降并不是由框架的复杂性引起的,在一次算法运行过程中需要包含以下阶段:①读取文件时间;②写入文件时间;③计算算法公式耗费的时间;④进程在 Rank0 和各个 Rank 之间传输的时间。各个部分所占时间如图4所示。

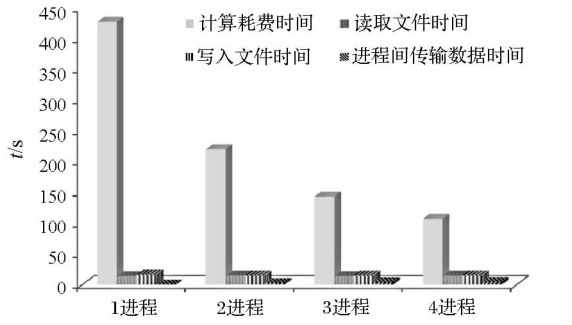

图4 运行时间对比图

Fig. 4 The comparison of each stage

从图4可以看出,随着进程数目的提高,运行算法的计算耗时一直呈明显下降趋势,说明通过框架程序编写傅里叶变换算法确实可以利用 MPI 技术降低算法运行时间。但是不管多少进程参与计算,其总体的读取文件时间、写入文件时间相对比较固定,随着总体运行时间的降低所占比例会不断提高;而进程间传输数据时间,在总体运行时间随着进程数的提高,占用时间和比例不断提高;这也就是为什么并行效率会随着进程数的增加而下降的原因。所以利用本框架程序提高算法运行速度,也需要考虑到磁盘 I/O、进程数据传输对算法效率的影响。通过以上实验可以充分证明本研究提出的框架可以有效地进行大型栅格数据的并行计算。

4 结论

本研究提出了一种面向大型地理栅格数据的并行处理框架,通过数据块划分、真实与虚拟读取、分块数据访问以及任务机制构造了程序框架结构。通过该框架,可以依据集群能力将栅格数据算法处理的过程划分为多个子任务,实现超大型数据的处理;将栅格处理算法与并行调度、磁盘 I/O 分离,实现模块的"高内聚低耦合"。实验表明,本研究提出的框架不但可以有效处理超过集群环境内存大小的数据,而且可以有效降低并行程序代码量和复杂度,提高软件质量。

参考文献(References)

[1] 宫鹏, 黎夏, 徐冰. 高分辨率影像解译理论与应用方法中的一些研究问题[J]. 遥感学报, 2006, 10(1):1 - 5.
GONG Peng, LI Xia, XU Bing. Interpretation theory and application method development for information extraction from high resolution remotely sensed data[J]. Journal of Remote Sensing, 2006, 10(1):1 - 5. (in Chinese)

[2] 刘军志, 朱阿兴, 刘永波, 等. 基于栅格分层的逐栅格汇流算法并行化研究[J]. 国防科技大学学报,2013, 35(1): 123 - 129.
LIU Junzhi, ZHU Axing, LIU Yongbo, et al. Parallelization of a grid-to-grid routing algorithm based on grids layering[J]. Journal of National University of Defense Technology, 2013, 35(1):123 - 129. (in Chinese)

[3] Plaza A, Valencia D, Plaza J, et al. Commodity cluster-based parallel processing of hyperspectral imagery[J]. Journal of Parallel and Distributed Computing, 2006, 66(3): 345 - 358.

[4] Plaza A, Du Q, Chang Y L, et al. High performance computing for hyperspectral remote sensing[J]. Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of,2011(99): 1 - 17.

[5] Hawick K A, Coddington P D, James H A. Distributed frameworks and parallel algorithms for processing large-scale geographic data[J]. Parallel Computing, 2003, 29(10): 1297 - 1333.

[6] Aloisio G, Cafaro M. A dynamic earth observation system[J]. Parallel Computing, 2003, 29(10): 1357 - 1362.

[7] Li X, Zhang X H, Yeh A, et al. Parallel cellular antomata for large-scale urban simulation using load-balancing techniques [J]. International Journal of Geographical information Science, 2010,24(6):803 - 820.

[8] 沈占锋, 骆剑承, 陈秋晓, 等, 基于 MPI 的遥感影像高效能并行处理方法研究[J]. 中国图象图形学报,2007, 12(12):2132 - 2137.
SHEN Zhanfeng, LUO Jiancheng, CHEN Qiuxiao, et al. High-efficiency remotely sensed image parallel processing method study based on MPI[J]. Journal of Image and Graphics, 2007, 12(12):2132 - 2137. (in Chinese)

[9] 黄国满, 郭建峰. 分布式并行遥感影像处理中的数据划分[J]. 遥感信息, 2001, (2): 9 - 12.
HUANG Guoman, GUO Jianfeng. Data Partition for Distributed-parallel Processing of remote sensing imagery[J]. Remote Sensing Information, 2001, (2): 9 - 12. (in Chinese)

[10] 朱志文,沈占锋,骆剑承. 改进 SIFT 点特征的并行遥感影像配准[J]. 遥感学报 2011,15(5),1032 - 1039.
ZHU Zhiweng, SHEN Zhanfeng, LUO Jiancheng. Parallel remote sensing image registration based on improved SIFT point feature[J]. Journal of Remote Sensing, 2011,15(5): 1032 - 1039. (in Chinese)

[11] Shen Z F, Luo J C, Zhou C H, et al. System design and implementation of digital-image processing using computational grids[J]. Computers & Geosciences,2005, 31(5): 619 - 630.

[12] 欧阳柳,熊伟,程果,等. 地理栅格数据的并行访问方法研究[J]. 计算机科学,2012,39(11):116 - 121.
OUYANG Liu, XIONG Wei, CHENG guo, et al. Parallel access methods for geographic raster data[J]. Computer Science, 2012,39(11):116 - 121. (in Chinese)

[13] Löscher A, et al. Variational optimization for global climate analysis on ESA's high performance computing grid[J]. Remote Sensing of Environment, 2008,112(4):1450 - 1463.