地理栅格数据的并行访问方法研究

欧阳柳 熊 伟 程 果 陈宏盛 陈 荦
(国防科技大学电子科学与工程学院 长沙 410073)

摘 要 在海量地理栅格数据处理中,数据 I/O 性能是影响处理算法程序整体性能的关键。目前针对地理栅格数据 I/O 优化问题的研究成果还很有限,通过对并行程序中的数据 I/O 模式进行深入分析,结合栅格数据逻辑模型和物理模型的特点,提出了面向地理栅格数据的并行 I/O 框架;基于消息传递模型,实现了 4 种并行访问方法。实验证明,并行访问方法优于传统的串行访问方法和分时多进程访问方法。该研究成果可以提高并行栅格处理程序的 I/O 访问效率,进而提高其整体并行性能。

关键词 地理栅格数据,并行数据访问,并行栅格数据处理,消息传递模型
中图法分类号 TP311 **文献标识码** A

Parallel Access Methods for Geographic Raster Data

OUYANG Liu XIONG Wei CHENG Guo CHEN Hong-sheng CHEN Luo
(Department of Electronic Science and Engineering, National University of Defense Technology, Changsha 410073, China)

Abstract In the field of geographic raster data processing, the data I/O performance significantly impacts the performance of a parallel program, especially when data are massive. Currently, the research on parallel optimizations for geographic raster data I/O is quite limited. By combining the data I/O paradigms in parallel programs with the characteristics of the geographic raster logical and physical models, we proposed a parallel I/O architecture for geographic raster data, and implemented four parallel access methods based on message passing model. Massive experiments verify that our methods outperform not only the conventional sequential I/O method but also the time-sharing multi-processes data access method. Our work can be used to promote the I/O efficiency of a parallel raster processing algorithm and thus improve the parallel performance of the program.

Keywords Geographicraster data, Parallel I/O, Parallel raster processing, Message passing model

1 引言

高性能计算已被公认为继理论科学和实验科学之后,人类认识世界、改造世界的第三大科学研究方法,是科技创新的重要手段[1]。并行计算是高性能计算领域中一类重要技术,通过将计算任务和/或计算数据合理划分并分配到多个计算单元上同时执行计算,实现算法程序的并行加速。随着近年来软硬件技术的飞速发展、多核处理器的日益普及,并行计算的门槛越来越低,它已被用于求解多个领域的复杂问题。在本文所关注的地理栅格数据处理方面,学术界乃至工业界已涌现了很多并行栅格处理方面的研究成果[2,3]。集中式 I/O[4]的方法是在执行访问和分发结果之前,在几个进程之间将请求聚集,以降低系统和磁盘的开销,从而实现 I/O 优化。许多学者对集中式 I/O 做了大量研究,它既可以在磁盘的层次上实现(Disk-directed I/O)[5],也可以在服务器端实现(Server-directed I/O)[6]。Rajeev Thakur 提出了一种扩展的两阶段式 I/O 方法[7],从而有效地提高了访问分布式文件的效率。

并行程序中计算部分的性能可以随着处理器性能和数量的增加而不断提高;相对而言,I/O 性能的增长跟不上计算性能的增长。在地理栅格数据处理领域,当面对大规模数据处理问题时,低性能的 I/O 将成为影响整体并行性能的瓶颈。D. A. Patterson 认为[8],目前计算机系统的性能应从传统的追求 CPU 计算能力(MIPS)的提高转向追求整体性能提高,特别是要提高系统 I/O 能力。在提高和改善 I/O 性能方面,国内外学者已经取得了一定成果。1998 年,美国超级计算中心推出了一种新型数据格式 HDF5(层次式文件格式),该格式具有自描述性、可扩展性、自组织性,可用于绝大多数科学研究的数据存储格式[9]。目前很多大型科学研究项目都采用 HDF5 作为统一的数据处理格式,如美国宇航局 NASA 的地

到稿日期:2012-05-29 返修日期:2012-07-02 本文受国家自然科学基金(61070035,60902036),高等学校博士学科点专项科研基金(20104307110017),国家高技术研究发展计划(863 计划)主题项目(2011AA120300)资助。
欧阳柳(1989-),女,硕士生,主要研究方向为并行计算、高性能计算、地理计算等,E-mail:oyangliu@mail.ustc.edu.cn;熊 伟(1976-),男,博士,副教授,主要研究方向为空间数据库、地理信息系统等;程 果(1984-),男,博士生,主要研究方向为高性能计算、空间信息处理;陈宏盛(1958-),男,硕士,副教授,主要研究方向为地理信息系统、数据库技术;陈 荦(1973-),男,博士,教授,主要研究方向为高性能计算、地理空间信息处理、地理信息系统等。

球科学数据与信息系统、美国百万亿次计算机 ASCI 计划的数据模型与格式组等[10]。为了更好地满足用户对科学数据管理的高效性需求，HDF5 支持多 I/O 机制，包括文件簇 I/O、并行 I/O 与网络 I/O 等。目前国家气象卫星接收的大部分数据存储格式也是 HDF 栅格图像，HDF5 格式的层次式为用户表达提供了最大的灵活性，这也是 HDF5 和传统遥感数据格式最大的不同之处。但是 HDF5 采用二叉树的方式建立文件内容的索引，相比于 TIFF 等传统数据格式，HDF5 的物理结构更为复杂。此外，开源数据模型 GDAL（Geospatial Data Access Library）[11]在 1.9 版本中已经支持并行数据读入功能。但 GDAL 的并行读取只是多进程读取技术，在数据访问过程中实际上是多进程轮流串行读取，是一种伪并行的数据访问方法。近年来，并行地理计算的研究成为热点，用并行计算技术解决地理学中的数据密集型和计算密集型问题已成为发展趋势。研究并行地理计算中的 I/O 问题可以有效地解决海量空间数据访问的瓶颈问题，进而提高并行地理程序的综合性能。目前针对地理栅格数据 I/O 问题的研究成果还很有限，在这种情形下，本文从提升并行地理计算整体性能的实际应用需求出发，以提高面向地理栅格数据的 I/O 性能为目的，研究并行 I/O 框架，构建逻辑映射模型，解析地理栅格元数据，分析对比多种并行访问模式，最终研发实现面向地理栅格数据的并行 I/O 方法。本文的研究成果可以被广泛地应用于并行地理计算，帮助地理学家研发高 I/O 性能的并行栅格处理算法程序。

2 并行程序中的数据 I/O 模式

数据 I/O 在并行程序中是必不可少的，且并行程序中的 I/O 与串行程序是有所不同的。并行程序的 I 指的是将数据从数据文件中读入至每个进程的内存中；而 O 指的是将数据从每个进程的内存写出至数据文件。

2.1 分发/收集式 I/O

传统的数据 I/O 模式一般是基于主从结构的分发/收集式。这种模式将某一进程选作为主进程，其他进程作为从进程负责计算。如图 1 所示，P_0 为主进程，P_1-P_n 为从进程。数据 I/O 的开始是由 P_0 从文件系统的数据文件中读入数据，进行数据划分后以消息传递的方式分发数据至 P_1-P_n；P_1-P_n 对自己分配到的数据完成计算处理后，再以消息传递的方式将数据收集至 P_0，并由 P_0 统一将数据写出至文件系统的数据文件。

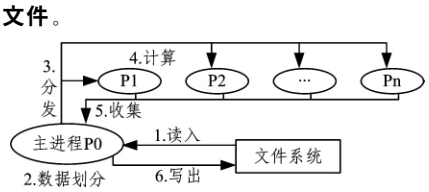


图 1 分发/收集式 I/O 流程

在这种模式下，并行程序的 I/O 性能取决于通信带宽和主进程的数据访问速度。这种方式最大的优点是不依赖于任何特定的软硬件环境，容易实现；缺点是数据访问时主进程的工作量大，数据分发/收集过程中通信开销大，容易造成通信阻塞。

2.2 并行 I/O

近年来，新型存储软硬件技术的发展与成熟有效地弥补了磁盘和主存之间速度的巨大差异，提高了计算机的整体性能。硬件方面，RAID（Redundant Arrays of Inexpensive Disk）[12]技术将多台磁盘构成磁盘阵列，对数据提供并行交叉存取，有效地提高了磁盘 I/O 速度，并利用冗余技术提高了可靠性；软件方面，PVFS[13]、GPFS[14]、Lustre[15]等并行文件系统可以充分利用并行计算机丰富的硬件资源，具有很好的并行吞吐性能。一方面，通过在底层物理存储设备、存储网络以及计算节点、I/O 服务节点等各层次中的虚拟化和并行化来提供高性能并发数据访问能力；另一方面，通常某种锁机制在实现高性能数据传输的同时保证数据的全局一致性[14]。

并行文件系统的上层是并行 I/O 库，指的是实现对并行文件系统中存储的数据文件进行并行 I/O 所需的软件支持库和程序接口集。目前最流行的并行 I/O 库是 MPI-IO，它作为一种规范，提供了执行可移植的、有效的 I/O 操作的接口，通过该接口可以在文件和进程间传送数据[16]。然而大多数并行 I/O 库接口繁琐、使用不便，本文的研究将基于并行 I/O 库，利用并行文件系统提供的并行 I/O 能力，研究面向地理栅格数据的并行 I/O 方法。

并行 I/O 指的是多个进程同时对同一共享文件实现数据访问。如图 2 所示，在并行 I/O 模式中，主进程 P_0 只是从数据文件中提取元数据信息，然后进行数据划分并把划分后对应于各个从进程 P_1-P_n 的元数据以消息传递的方式分发。P_1-P_n 通过参考元数据直接对并行文件系统中的数据文件提出读请求，读入相应的子数据集。在完成计算后，P_1-P_n 再次参考元数据将结果直接写入并行文件系统中的数据文件。相比于分发集中模式，并行 I/O 模式避免了主进程独自读取大规模数据，也消除了主从进程间的通信拥挤现象，只要并行文件下系统具有不错的并行吞吐力，并行 I/O 就一定会显著地提升程序的 I/O 性能。

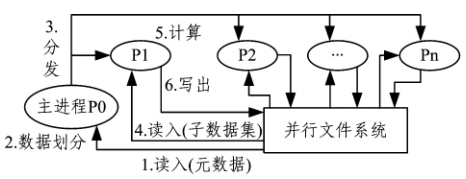


图 2 并行 I/O 流程

3 面向地理栅格数据的并行 I/O

地理栅格数据一般采用矩阵形式存储，所以对于栅格数据的处理，可以借鉴科学计算中的矩阵处理。但是地理栅格数据的结构更为复杂，不仅包括栅格图像数据，还包括地理元数据等信息，且存在多波段现象，因此面向地理栅格数据的并行访问技术更为复杂。

3.1 栅格逻辑结构向一维物理结构的映射

对于指定宽 w、高 h 的某地理栅格数据，在逻辑上它是一个 $w\times h$ 的二维矩阵结构，而在物理上它是一个长度为 $w\cdot h$ 的一维数组结构。地理栅格数据一般采用行优先存储方法，即将二维矩阵中的每个点数据按照行顺序映射为一维数组。由图 3 可知，假定矩阵的左上角坐标为(0,0)，那么对于任意给定点(x_0,y_0)，该点映射到一维数组中的物理偏移量 $f(x_0,y_0)$的计算公式为：

$$f(x_0,y_0)=(x_0\cdot w+y_0)\cdot sizeof(pixel) \quad (1)$$

式中，$sizeof(pixel)$表示每个像素点所占的字节数。

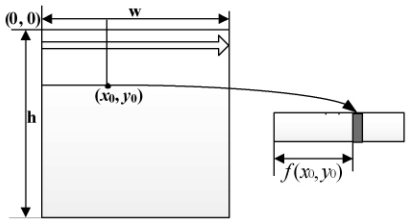


图 3　二维逻辑结构向一维物理结构的映射

而地理栅格文件的实际存储结构中还涉及元数据存储，但同一波段的点数据一般是连续存储的。假设 $offset$ 为上述起始点(0,0)相对于文件起始位置的物理偏移量，则公式(1)应进一步修改为：

$$f(x_0, y_0) = offset + (x_0 \cdot w + y_0) \cdot sizeof(pixel) \quad (2)$$

在并行栅格数据处理程序中，需要通过数据划分将大规模栅格数据划分生成多个小规模数据块，每个进程处理一个数据块。假设参照图 4，将栅格数据划分成 $m \cdot n$ 个数据块，那么对于进程 P_x，它分配处理的数据块实际上是一系列点的集合。而这些点从二维矩阵映射到一维数组后对应的物理偏移量的集合式为：

$$S(P_x) = \{ f(x, y) \mid x \in [x_0, x_0 + \frac{h}{n}], y \in [y_0, y_0 + \frac{w}{m}] \} \quad (3)$$

式中，x_0 和 y_0 可以根据 P_x 进程所获取的分块的行列序号计算得出。

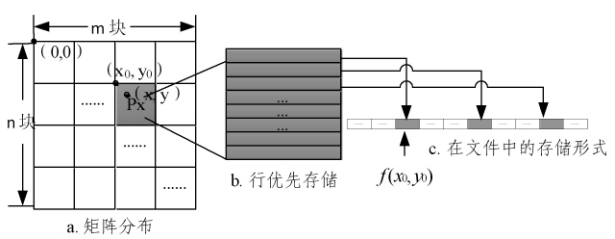


图 4　数据划分后每个数据块点的偏移量集合

借助式(2)、式(3)，即可计算出任意进程 P_x 需要处理的点在数据文件中的物理偏移量，从而能够实现上节中并行 I/O 方法提到的直接对文件实施数据读写操作。参考式(2)、式(3)可以看出，将矩阵中的逻辑位置映射到文件物理位置的计算过程中需要用到栅格数据的宽 w、高 h 以及数据相对于文件起始处的偏移量 $offset$ 等元数据信息。所以并行访问数据的基础是要对地理栅格数据进行解析，提取上述元数据信息。本文的研究对象选择了目前比较常用的标记图像格式(Tagged Image File Format，TIFF)文件。

TIFF 格式具有很好的可扩展性和灵活性，已被广泛应用于数字影像、遥感、医学等领域。TIFF 的数据结构以 8 字节长的图像文件头(Image File Header，IFH)开始，文件头中最重要的成员是一个指向名为图像文件目录(Image File Directory，IFD)的数据结构的指针。IFD 是 TIFF 文件中最重要的数据结构，每个 IFD 都标识了一幅图像的基本属性，而一个 DE(Directory Entry)就是一幅图像的某一个属性。如果一个 TIFF 文件包含多幅图像，就会有多个 IFD。当前 TIFF 6.0 标准手册中为 TIFF 文件定义了 70 多种不同类型的标记(Tag)，有的用来存放以像素为单位的图像宽度和高度，有的用来存放色表(如果需要的话)，有的用来存放位图数据的标记等等。本文需要用到的几个元数据标记如表 1 所列。

表 1　DE(Directory Entry)中 Tag 说明

Tag	属性说明	Type
256	图像宽	Integer
257	图像高	Integer
273	图像数据起始字节相对于文件开始处的位置	Long
279	图像数据字节总数	Integer
…	…	…

3.2　面向地理栅格数据的并行 I/O 框架

本节将结合 2.2 节提出的并行 I/O 模式、3.1 节给出的映射公式和 3.2 节展示的 TIFF 解析方法，构建面向 TIFF 格式地理栅格数据的并行 I/O 框架。

在 TIFF 文件中，IFD 数据单元并不一定紧跟在 IFH 数据单元后面，相反，它常常位于图像数据单元之后。也就是说，TIFF 格式的栅格数据一般组织形式是：IFH-栅格数据-IFD。下面参照图 5，详述本文提出的面向 TIFF 格式地理栅格数据的并行 I/O 框架。

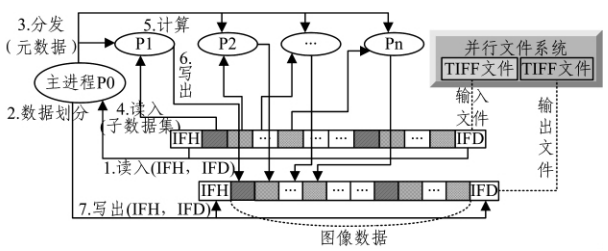


图 5　面向 TIFF 格式栅格数据的并行 I/O 框架

数据 I/O 的开始是由主进程 P_0 解析 TIFF 文件，获取 IFH 和 IFD，进而从中提取出关键的元数据信息(包括栅格数据的宽、高、每个数据点的数据类型、偏移量等信息)；而后依然由 P_0 根据这些信息对地理栅格数据进行数据划分，并把对应于各个从进程 $P_1 - P_n$ 的元数据进行分发；$P_1 - P_n$ 接收到元数据信息后，参考式(2)、式(3)计算自己负责计算的数据块的物理偏移量集合 $S(P_x)$，而后直接访问并行文件系统中的文件，通过偏移指针读取需要的数据；在 $P_1 - P_n$ 执行并行计算时，P_0 负责在并行文件系统中建立一个新的结果文件；当并行计算完成后，$P_1 - P_n$ 将计算后的数据按照刚才计算所得的物理偏移量写入到结果文件中，而 P_0 则负责将文件头以及其他元数据信息(即 IFH、IFD)写入至结果文件。这样才能完成一个完整 TIFF 栅格数据文件的写出操作。

3.3　4 种并行数据访问方法

在实现上述地理栅格数据并行 I/O 框架时，采用了 MPI-IO 提供的并行访问接口。在实现过程中，通过深入分析地理栅格数据的特点，结合使用 MPI-IO 提供的接口，一共总结归纳出了 4 种并行数据访问方法。本小节将分别介绍每种方法的原理、特点，分析其可能的适用条件，在实际应用中选择最合适的方法以最大限度地提升数据 I/O 性能。

如图 6 所示，4 种方法分别被定义为 level 1－level 4。

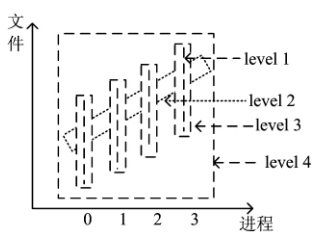


图 6　4 种并行数据访问方法原理示意

level 1 的核心思想是每个进程对自己负责计算的数据块的每一行执行独立的 I/O 操作。每个进程首先根据公式计算出数据块起始点(0,0)映射的物理偏移量,读/写第一行数据,然后计算下一行的物理偏移量,读/写第二行数据,依此类推。level 1 进行数据访问时,每个进程的 I/O 操作数等同于数据块的行数。如果栅格数据的规模非常大,每个进程就需要访问大量非连续的小数据块,从而产生大量 I/O 请求,影响 I/O 性能。

level 2 的大体思想与 level 1 类似,只不过 level 2 借鉴了集中式 I/O[4] 的思想,在执行访问和分发结果之前,在几个进程之间将请求聚集,因此,相比于 level 1,level 2 对文件的I/O 请求次数较少。但是聚集操作会增加各进程之间的等待时间,一旦进程数量过多,其 I/O 性能就会受到影响。

level 3 的核心思想是利用抽象数据结构对每个进程设置一个控制该进程数据访问范围的虚拟化描述(称为文件视图),使各个进程访问其数据块时,虽然数据在物理上是离散存储的,但其在逻辑上是连续的。这样一来,level 3 在并行访问数据块时,文件视图将同一进程的多次 I/O 操作合并为一次 I/O 操作,I/O 的数据量等于该进程的视图大小。所有进程对文件的 I/O 操作总数等于进程数。level 3 的优点是增加了数据访问的灵活性,减少了 I/O 操作次数,从而降低了 I/O 开销。

level 4 的大体思想与 level 3 类似,只不过在 level 3 的基础上增加使用了集中式 I/O 思想。因此,level 4 进一步将 level 3 中所有进程的 I/O 操作合并为一个大数据量的 I/O 操作,其性能应该会进一步得到改善。

综上可见,level 1－level 4 这 4 种不同的方法具有不同的思想,因此也相应地适用于不同情况的并行数据读取。下面将通过大量对比性实验验证本文所提的并行 I/O 方法的有效性和高效性,分析总结上述 4 种方法的适用条件。

4 实验与结果分析

4.1 实验环境及说明

为了验证本文所提方法的有效性,基于某 IBM 高性能计算集群和不同规模的 TIFF 数据实施了一系列对比性实验。实验的软硬件环境及实验数据集的详细信息分别参见表 2－表 4。

表 2 硬件信息

硬件	配置细节
主控节点	x3650M3:2× Intel Xeon 4C X5540 2.4GHz,6×4GB DDR3 LP RDIMM
I/O 控制节点	4×x3550M3:2× Intel Xeon 4C X5570 2.80GHz,6×2GB DDR3 LP RDIMM
计算节点	28×Blade HS22:2× Intel Xeon 4C X5570 2.80GHz,6×2GB DDR3 LP RDIMM
网络	Infiniband
共享文件系统	GPFS

表 3 软件信息

软件	配置细节
操作系统	RedHat Enterprise Server 5.0
编译器	GCC 4.1.2
MPI	OpenMPI 1.4.3
其他库	GDAL 1.8.0

表 4 实验数据集信息

数据集	数据大小	数据规模
数据集 1	50MB	5120×10240
数据集 2	300MB	15360×20480
数据集 3	900MB	30720×30720
数据集 4	1.56GB	40960×40960

所有的实验都是在统一的集群环境下运行的。测试结果均为 10 次测量的平均值。以下所有实验结果图中,横坐标轴为程序并行运行的进程数 n,即当前使用的计算核心数,当 $n=1$ 时,表示是串行程序的测试;$n>1$ 时,是并行程序开启相应进程数的测试。我们实验采用的进程数是 2 的整数幂,即 $n=2^k, k\in\mathbb{N}$。纵坐标轴为可表示程序串行运行时间 t_1、并行运行时间 t_n,以及根据 t_1、t_n 和 n 计算而得的并行加速比 Sp 和并行效率 E。相应的计算公式如下:

$$Sp=\frac{t_1}{t_n} \tag{4}$$

$$E=\frac{Sp}{n}=\frac{t_1}{n\cdot t_n} \tag{5}$$

4.2 并行 I/O 实验

本文首先进行了面向 TIFF 格式栅格数据的并行 I/O 实验,并在实验中实现了全部 4 种并行 I/O 模式。本实验采用的数据划分方法为传统的行划分方法。如图 7(a)所示,栅格数据被行划分成等同于进程数量的分块,每个进程负责一个分块的 I/O。每个分块都拥有相同的行数,因此分块的行数 $sub\_row=h/n$,其中 h 为原始栅格数据的高,n 为进程数。

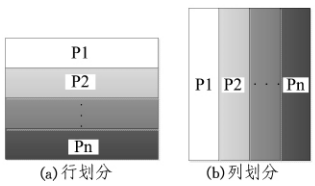


图 7 两种数据划分方法

接下来,以表 4 所列的数据集 2 作为测试数据,以行划分作为数据划分方式,在集群环境上分别测试了数据串行读取时间($n=1$)和使用 4 种并行数据访问方法(level 1－4)在不同数目进程下的数据并行读取时间。实验结果如图 8 所示,实验结论如下。

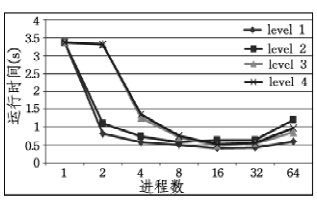


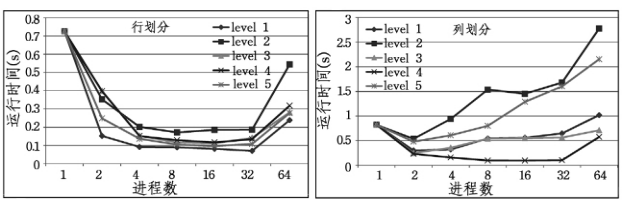
图 8 4 种访问方法在不同进程数时的并行数据读取时间

1. $n=1$ 时,表示只用一个进程串行地完成对栅格数据的读取操作,此时程序运行时间最长。$n\geqslant 2$ 后,表示用多个进程执行并行读取操作。随着 n 的增加,4 种访问方法的并行数据读取时间都在逐渐减小,说明了本文提出的并行 I/O 方法是有效的,可以缩短数据 I/O 时间,进而提升并行程序的整体性能。

2. 4 种访问方法对应的数据读取时间有着大体相同的走势,都是随着 n 增加至 32 后达到最低点,而后又逐渐上升。其原因为:(1)实验采用的集群拥有 4 个 I/O 节点,共 32 个核

心，因此当 $n \leqslant 32$ 时，随着 n 的增加，每个处理器需要的数据分块大小不断减小，使得读取时间相应下降；(2)而当 $n > 32$ 时，没有足够多的 I/O 核心提供并行 I/O 的支持，虽然分块更小，但是由于并行 I/O 速度也变慢，造成整体的读取时间不再减少，甚至逐渐上升。

为了比较本文所提并行 I/O 方法与 GDAL 并行数据读取方法的性能，在上述实验基础上进一步实现了基于 GDAL 的并行数据读取实验，并将其命名为 level 5。GDAL 包括了缓存优化、分块优化等机制，但其性能还是客观的。实验结果如图 9(a)所示，实验结论如下。



(a) 行划分　　(b) 列划分

图 9　5 种访问模式的并行读取时间

通过对比 5 种访问方法的并行数据读取时间，发现 level 5(即基于 GDAL 的伪并行)的读取时间很短，其性能优于除 level 1 以外的其他 3 个模式。这种结果并不代表我们提出的方法不可行，而是因为行划分时，每个进程处理的数据在文件中连续存储，如图 7(a)所示。对于这种物理上连续存储的数据读取，本文提出的并行 I/O 方法并无突出优势，而 GDAL 作为一个成熟的开源地理栅格数据访问库，它的实现包含多种优化策略。因此，GDAL 虽然是伪并行读取，但是拥有可观的性能。不过，相比之下，其性能仍然落后于本文提出的 level 1。

为了验证上述分析结论，我们对于同一数据集，编码实现了列划分方法(分块的列数 $sub\_col = w/n$，见图 7(b))，并重新测试了 5 种方法的并行数据读取时间。实验结果如图 9(b)所示，实验结论如下。

1. level 1、level3、level 4 性能都要优于 level 5。这是因为列划分时每个进程需要处理的数据块在文件中非连续存储，当采用 level 5 模式访问数据时，每个进程先定位到文件中的适当位置，读取一行数据，然后定位到下一个位置，读取下一行，依此类推。所以列划分情况下，使用 level 5 的并行程序中各进程的寻址时间会大大增加，这种基于 GDAL 实现的伪并行 I/O，无法支持多进程多指针并行读取文件，所以性能随着进程数的增加而大幅度下降。

2. 对于 level 1、level 2、level 3 和 level 5，随着进程数的增加，这 4 种方法的性能也会有所下降。这是因为进程数的增加使得矩阵划分的列数增加，每个进程获得的数据块大小减小，但是列划分情况下每个进程的 I/O 请求数不会变化，所以总的数据读取时间会增加，但是增加得很缓慢。

3. 在所有 5 种方法中，level 2 的性能最差。这是因为 level 2 使用了聚集读取思想，每次执行该函数时，需要等待其他进程执行相同的函数来进行 I/O 请求，每次等待后的 I/O 请求数据量就是所有进程的子矩阵的一行，这样可能会使得进程之间等待的开销大于每个进程单独请求子矩阵每一行的开销，从而影响 level 2 的性能。

4. level 3 为每个进程设置视图，使得每个进程一次 I/O 请求的数据量是该进程中视图的大小，对文件的 I/O 次数就等于进程数，相比于 level 2 和 level 1 减少了对文件的 I/O 次数，所以性能有所提高。

5. level 4 的性能最好。因为 level 4 在设置视图的基础上又使用聚集读取思想并行读取视图，即一次 I/O 操作请求的数据量是所有进程的所有视图的大小，请求的数据量也较大。而 level 3 每次请求的数据量只是每个进程的视图大小，并且所有进程都需要单独发出请求来访问本视图的数据。所以在并行程序中使用 level 4 访问数据时，I/O 性能最佳。

最后，为了证明数据集规模不会影响上述实验结论，我们基于 4 种不同规模的数据集(如表 4)实施了 level 4 和 level 5 的性能对比实验。实验结果如图 10 所示，实验结论如下。

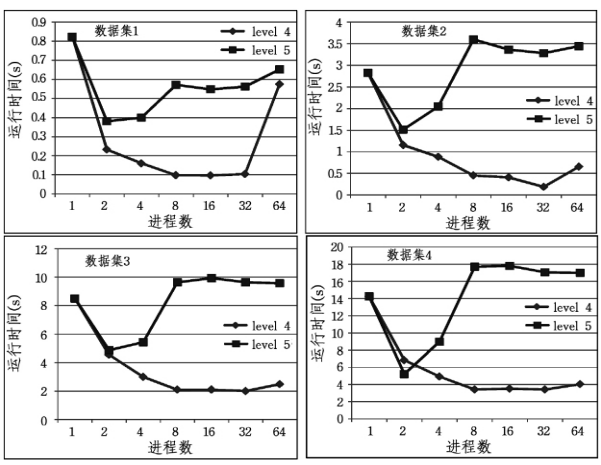
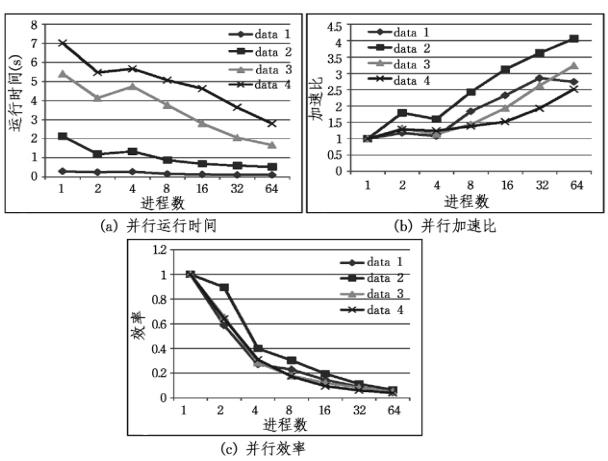


图 10　level 4 与 level 5 性能对比

在数据按列划分情况下，对于任意规模的数据集，本文提出的 level 4 的性能都比基于 GDAL 的 level 5 要好，且数据集规模越大，性能相差越明显。

总之，本文设计实现的 level 4 模式的并行 I/O 方法的效果最好，其并行数据读取的时间值、并行加速比和并行效率如图 11 所示。



(a) 并行运行时间　　(b) 并行加速比

(c) 并行效率

图 11　level 4 对于不同数据集下的并行读取性能

结束语　数据 I/O 已成为并行栅格数据处理算法中影响整体并行性能的关键问题，研究地理栅格数据提出的并行访问方法具有十分重要的应用价值。本文分析了并行程序中的两种数据 I/O 模式，提出了面向地理栅格数据(以 TIFF 格式为代表)的并行 I/O 框架和访问方法。实验结果验证了本文

方法的有效性和高效性。本文的研究成果可以应用于并行栅格数据处理算法的研究，以提高 I/O 性能。

下一步将着手完善本文所提的 4 个方法，增加多波段支持和十字型数据划分方法；借鉴 GDAL 的缓存管理等优化策略，进一步优化并行 I/O 性能；在此基础上，增加更多栅格数据格式的支持，最终研发一个高性能的并行 I/O 类库，并将其应用于并行栅格处理程序中的 I/O 操作。

参 考 文 献

[1] 周毓麟，沈隆钧. 高性能计算的应用及战略地位[J]. 中国科学院院刊，1999(3)：184-187

[2] Guan Q，Clarke K C. A general-purpose parallel raster processing programming library test application using a geographic cellular automata model[J]. International Journal of Geographical Information Science，2010，24(5)：695-722

[3] Cheng G，Liu L，Jing N，et al. General-purpose optimization methods for parallelization of digital terrain analysis based on cellular automata[J]. Computers & Geosciences，2012，45：57-67

[4] Nitzberg，Bill，Lo V. Collective Buffering：Improving Parallel I/O Performance[C]// Proceedings of the Sixth IEEE International Symposium on High Performance Distrubuted Computing. 1997，8：148-157

[5] Kotz D. Disk-directed I/O for MIMD Multiprocessors[J]. ACM Transactions on Computer Systems，1997，15(1)：41-47

[6] Seamons K，Chen Y，Jones P，et al. Server-Directed Collective I/O in Panda[C]// Processdings of Supercomputing. ACM Press，December 1995

[7] Thakur，Rajeev. An Extended Two-phase Method for Accessing Sections of out-of-core Arrays[J]. Scientific Proramming，1996，5(4)：301-317

[8] Gibson G A，Vitter J S，Wilkes J. Report of the Working Group on Storage I/O Issues in Large-Scale Computing[J]. ACM Computing Surveys，1996，28(4)：12-28

[9] Hierarchical Data Format Group National Center for Super_computing Applications. HDF5 Data Model[Z]. University of Illinois at Urbana-Champaign，Sep. 2004

[10] HDF 5 Users. Hierarchical Data Format Croup National Center for Super_computing Applications University of Illinois at Urbana-Champaign[OL]. http://hdf. ncsa. uiuc. edu/users5. html，May 2004

[11] Warmerdam F. The geospatial data abstraction library open source approaches in spatial data handling[M]// Hall G B，Leahy M G，eds. Springer Berlin Heidelberg，2008：87-104

[12] Overview of I/O Performance and RAID in an RDBMS Environment [EB/OL]. http://www. perftuning. com/whitepapers/RAID. pdf，2002-06

[13] Parallel Virtual File System[EB/OL]. http://www. pvfs. org

[14] Schmuck F，Haskin R. Gpfs：A shared-disk file system for large computing clustersed[C]// Proceedings of the Conference on File and Storage Technologies，USENIX Association. 2002：231-244

[15] Lustre Cluster File System[EB/OL]. http://www. luster. org

[16] The MPI-IO Committee. MPI-IO：A Parallel File I/O Interface for MPI，Version 0. 5. World-Wide Web[OL]. http://lovelace. nas. nasa. gov/MPI-IO，April 1996

（上接第 85 页）

[2] 刘玉英，史旺旺. 一种基于遗传算法的无线传感器网络节点优化方法[J]. 传感器学报，2009，22(6)：869-872

[3] 袁浩. 基于改进蜂群算法无线传感器感知节点部署优化[J]. 计算机应用研究，2010，27(7)：2704-2705

[4] 崔莉，鞠海玲，苗勇，等. 无线传感器网络的研究进展[J]. 计算机研究与发展，2005，42(1)：163-174

[5] 刘丽萍，王智，孙优贤. 无线传感器网络中的资源优化[J]. 传感技术学报，2006，19(3)：918-919

[6] 李晓磊，路飞，田国会，等. 组合优化问题的人工鱼群算法应用[J]. 山东大学学报：工学版，2004(05)：64-67

[7] 班晓娟，彭立，王晓红，等. 人工鱼群高级行为的自组织算法与实现[J]. 计算机科学，2007(07)：193-196

[8] 亚湘，孙文瑜. 最优化理论与方法[M]. 北京：北京科技出版社，1997

[9] 王永才，赵千川，郑大钟. 传感器网络中能量最优化的聚类轮换算法[J]. 控制与决策，2006，4：400-404

[10] 李莉，刘元安，唐碧华. 一种基于网格模型的传感器节点放置算法[J]. 武汉大学学报：理学版，2007(S)：83-87

[11] 赵国炳，陈国定，张奇伟. 一种无线传感器网络覆盖优化方法[J]. 机电工程，2009(06)：80-82

[12] 无线传感器网络节点部署问题研究[J]. 计算机学报，2007，30(4)：563-568

[13] 王联国，洪毅，赵付青，等. 一种改进的人工鱼群算法[J]. 计算机工程，2008(19)

[14] Huang C F. The coverage problem in wireless sensor network [C]// Proc of ACMInternational Workshop on Wireless Sensor Networks and Applications. New York：ACMPress，2005：519-528

[15] Jiang Y，Wang Y，Pfletschinger S，et al. Optimal multiuser detection with artificial fish swarmalgorithm[C]// Proc of International Conference on Intelligent Computing. Berlin，Heidelberg：Springer-Verlag，2007：1084-1093

[16] Bonabeau E，Theraulaz G. Swarm smarts[J]. Scientific American，2000，282(3)：72-79

[17] Jonathan H，Zhiyuan R，Bruce H. Sentry-Based Power Management in Wireless Sensor Networks[C]// IPSN 2003. 2003：458-472

[18] Lu Jun，Tatsuya S. Coverage-aware self-scheduling in sensor networks[C]// Proceedings of IEEE 18th Annual Workshop on Computer Communications(CCW 2003). 2003，10：117-123

[19] Wu K，Gao Y，Li F. Light weight deployment-aware scheduling for wireless sensor networks[J]. Mobile Networks and Applications，2005，10(6)：837-852

[20] Suganthan P N. Particle swarm optimizer with neighbourhood operator[C]// Proceedings of the 1999 Congresson Evolutionary Computation. 1999：1958-1962

[21] Jourdan D B，de Weck O L. Layout optimization for a wireless sensor network using a multi-objective genetic algorithm[C]// IEEE 59th Vehicular Technology Conference (VTC 2004-Spring). Vol. 5，2004：2466-2470