Transactions in GIS

How to Apply the Geospatial Data Abstraction Library (GDAL) Properly to Parallel Geospatial Raster I/O?

Cheng-Zhi Qin,* Li-Jun Zhan* and A-Xing Zhu*†

*LREIS, IGSNRR, CAS, Beijing
†Department of Geography, University of Wisconsin-Madison

Abstract
Input/output (I/O) of geospatial raster data often becomes the bottleneck of parallel geospatial processing due to the large data size and diverse formats of raster data. The open-source Geospatial Data Abstraction Library (GDAL), which has been widely used to access diverse formats of geospatial raster data, has been applied recently to parallel geospatial raster processing. This article first explores the efficiency and feasibility of parallel raster I/O using GDAL under three common ways of domain decomposition: row-wise, column-wise, and block-wise. Experimental results show that parallel raster I/O using GDAL under column-wise or block-wise domain decomposition is highly inefficient and cannot achieve correct output, although GDAL performs well under row-wise domain decomposition. The reasons for this problem with GDAL are then analyzed and a two-phase I/O strategy is proposed, designed to overcome this problem. A data redistribution module based on the proposed I/O strategy is implemented for GDAL using a message-passing-interface (MPI) programming model. Experimental results show that the data redistribution module is effective.

1 Introduction

With the rapid growth of affordable parallel devices, parallel geospatial raster processing has become more widely used in dealing with increasingly massive geospatial raster datasets (e.g. Guan and Clarke 2010; Qin and Zhan 2012; Maulik and Sarkar 2012). However, two challenges still generally exist in the input/output (I/O) part of parallel geospatial raster processing. The first is the massive size of raster data, whose movement between fast main memory and slow disk storage often becomes the performance bottleneck of parallel geospatial raster processing. The second challenge is the diversity of geospatial raster data formats. Although there are presently at least several dozen often-used raster file formats, existing parallel geospatial raster-processing programs often support only a very few file formats, which limits their practical applicability.

Some studies have proposed to address the challenge of massive raster data by providing a parallel API to access a single file storing massive raster data in a specific format (e.g. Hierarchical Data Format v5 [HDF5], Parallel Network Common Data Format [PnetCDF], and Parallel Input-Output System [PIOS]) (Shook and Wang 2011). These studies have mainly focused on I/O performance while paying little attention to the challenge of diverse geospatial raster data formats.

The Geospatial Data Abstraction Library (GDAL, http://www.gdal.org/, current version number 1.9.2) (Warmerdam 2008), which is an open-source tool providing a single abstract

Address for correspondence: Cheng-Zhi Qin, State Key Laboratory of Resources and Environmental Information System, Institute of Geographical Sciences and Natural Resources Research, CAS, Beijing 100101, China. E-mail: qincz@lreis.ac.cn
Acknowledgements: This study was supported by the National High-Tech Research and Development Program of China (No. 2011AA120302), and the Institute of Geographic Sciences and Natural Resources Research, CAS (No. 2011RC203).

doi: 10.1111/tgis.12068

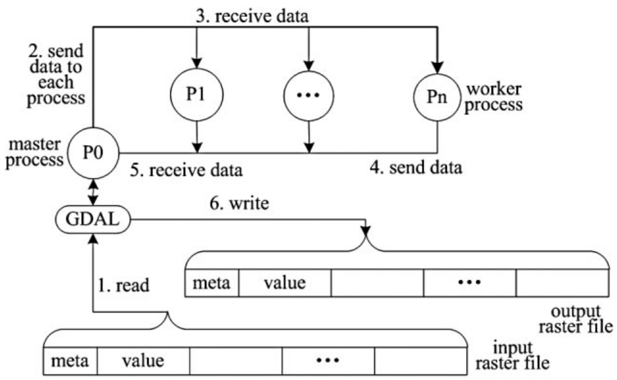

Figure 1 Serial raster I/O mode of applying GDAL to parallel geospatial processing

data model to read and write a variety of geospatial raster data formats, has been widely used in serial raster-processing applications. Some recent studies (e.g. Wang et al. 2012) have applied GDAL to parallel geospatial raster processing in a serial I/O mode for assessing raster data, i.e. a master process takes charge of the entire I/O process between external and internal memory, and other work processes access the data by communicating with the master process (Figure 1). However, this approach lacks parallelism and often creates a bottleneck when the size of the raster file exceeds the memory capacity of a single compute node.

Parallel raster I/O mode permits each process to access directly a part of the data stored in external memory, based on a kind of domain decomposition in which the data domain being processed is decomposed into subdomains. There are three commonly used, straightforward ways of domain decomposition: row-, column-, and block-wise. The parallel raster I/O mode could overcome the single-bottleneck problem in the serial raster I/O mode. However, there are few literatures on using GDAL in parallel I/O mode.

This article first explores the efficiency and flexibility of using GDAL in parallel raster I/O mode under the three common ways of domain decomposition. Experimentation shows that parallel raster I/O using GDAL cannot work well under two among the three ways of domain decomposition. Then a solution to this problem is proposed.

2 Using GDAL in Parallel Raster I/O Mode

When using GDAL in parallel raster I/O mode (Figure 2), the master process first uses GDAL to extract the metadata (e.g. spatial extent, projection) from a raster file and creates a corresponding empty output file. Then, following a specific domain decomposition approach, the master process sends the spatial extent information of each subdomain to the corresponding work process. Based on the subdomain information received, each process uses GDAL to open the shared input raster file and to read the subdomain data using the *RasterIO()* function in GDAL. After computation, each process uses GDAL to open the shared output raster file to write the results using the *RasterIO()* function. This parallel raster I/O mode could avoid certain problems in serial I/O mode, such as the single-bottleneck problem and the overhead for data distribution between the master process and the work processes.

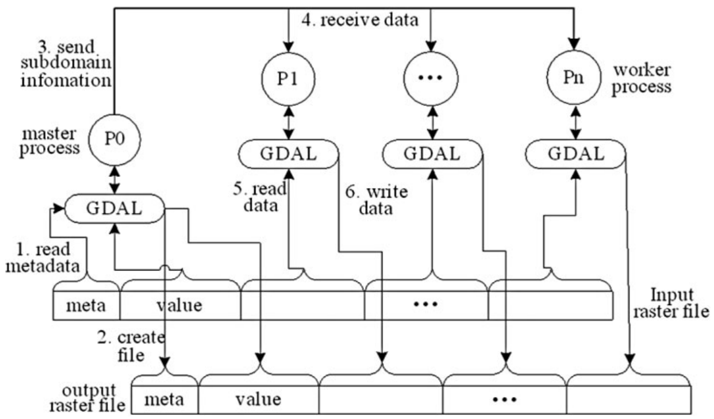

Figure 2 Parallel raster I/O mode of using GDAL for parallel geospatial processing

3 Problem in Parallel Raster I/O Using GDAL

In this section, the problem in parallel raster I/O using GDAL will be revealed by an experiment.

3.1 Experimental Design

Based on the message passing interface (MPI) programming model, the parallel raster I/O mode of using GDAL (called *GDAL_PIO*) was implemented. An experiment was designed to evaluate the efficiency and flexibility of *GDAL_PIO*. In this experiment, there is actually no computation, only read and write operations on the raster file, because this study concerns only I/O issues.

GDAL_PIO was tested on an IBM SMP cluster with 140 server nodes (including 6 I/O server nodes and 134 compute nodes). Each server node consists of two Intel Xeon (E5650 2.0 GHz) six-core CPUs and 24 GB DDRIII memory. The test raster data have a dimension of 23,300 × 27,825 cells. The data are stored as a file in GeoTiff format on I/O server nodes and shared between compute nodes through the General Parallel File System (GPFS).

Efficiency is evaluated by measuring the runtime to read and write data from/to a raster file with a specific format using *GDAL_PIO* (called I/O time). The time to create the output file is not counted in I/O time. Flexibility is evaluated by determining whether *GDAL_PIO* can work well under different domain decomposition approaches (i.e. row-, column-, and block-wise).

3.2 Experimental Results

The experimental results show that *GDAL_PIO* with row-wise domain decomposition is efficient (Figure 3). The I/O time using *GDAL_PIO* decreases when the number of processes increases from one to eight. The I/O time for *GDAL_PIO* with eight processes is about half that with one process. The I/O time for *GDAL_PIO* with 16 processes is longer than with eight processes, but is still shorter than with four processes.

However, using *GDAL_PIO* with column- or block-wise decomposition is highly inefficient, that is, almost 5–15 times slower than using *GDAL_PIO* with row-wise decomposition

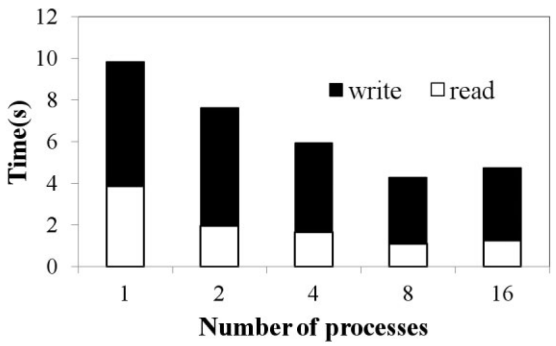

Figure 3 Runtimes for read and write operations of *GDAL_PIO* using row-wise decomposition when the number of processes varies from 1 to 16 (only one process per compute node)

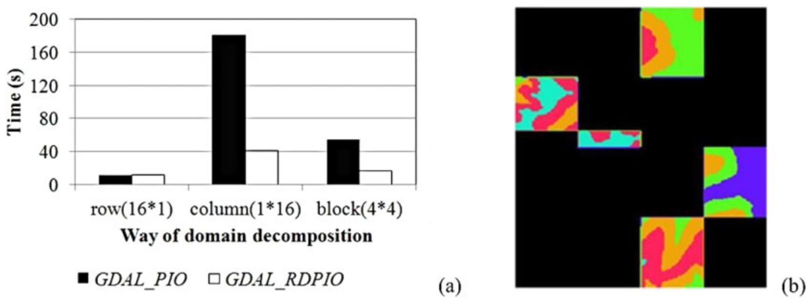

Figure 4 Performance of *GDAL_PIO* and *GDAL_RDPIO* (an implementation of the new parallel raster I/O strategy using GDAL proposed in this article, described later) executed on a cluster with a test raster file in GeoTiff format: (a) I/O times using *GDAL_PIO* and *GDAL_RDPIO* under three ways of domain decomposition (16 processes created on 16 compute nodes); and (b) example of incorrect results from *GDAL_PIO* with block-wise decomposition (black areas are blocks where results were missed)

(Figure 4a). Moreover, *GDAL_PIO* with column- or block-wise decomposition produced incorrect results in which some areas with values were output as having no data (the black holes in Figure 4b). The results for *GDAL_PIO* with row-wise decomposition were correct. Therefore, the experimental results show that *GDAL_PIO* lacks flexibility.

3.3 The Reason for the Problem in Parallel Raster I/O Using GDAL

This poor efficiency of *GDAL_PIO* with column- and block-wise decomposition is attributable to the fact that the processes are accessing several non-contiguous sections of a shared raster file. From the viewpoint of the user, a raster data file is a two-dimensional array. Therefore, when *GDAL_PIO* is executed with column- or block-wise decomposition, each process calls the *RasterIO()* function in GDAL only once to access data, similar to the situation with row-wise decomposition (Figure 5a). However, the raster data saved in disk are actually re-organized row-by-row as a contiguous linear stream of values which starts from the upper left corner cell in the raster array and ends with the lower right corner cell (Figure 5b). As a result, each process must access several non-contiguous sections of the file when *GDAL_PIO*

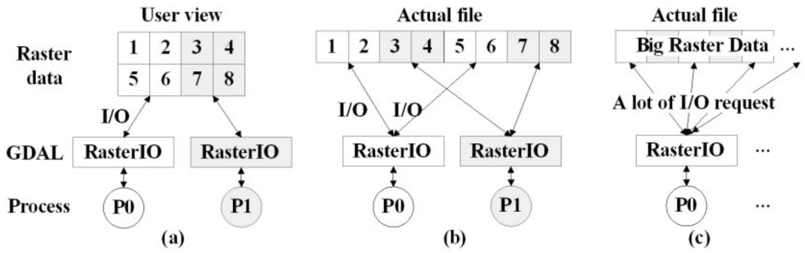

Figure 5 Analysis of the reason for the poor efficiency of *GDAL_PIO* with column-wise and block-wise decomposition: (a) I/O requests from only two processes using GDAL to access a 2 × 4 raster array with column-wise decomposition (from the viewpoint of the user); (b) the actual I/O requests in case a); and (c) a large number of small I/O requests by each process with column-wise decomposition of a large raster data array in reality

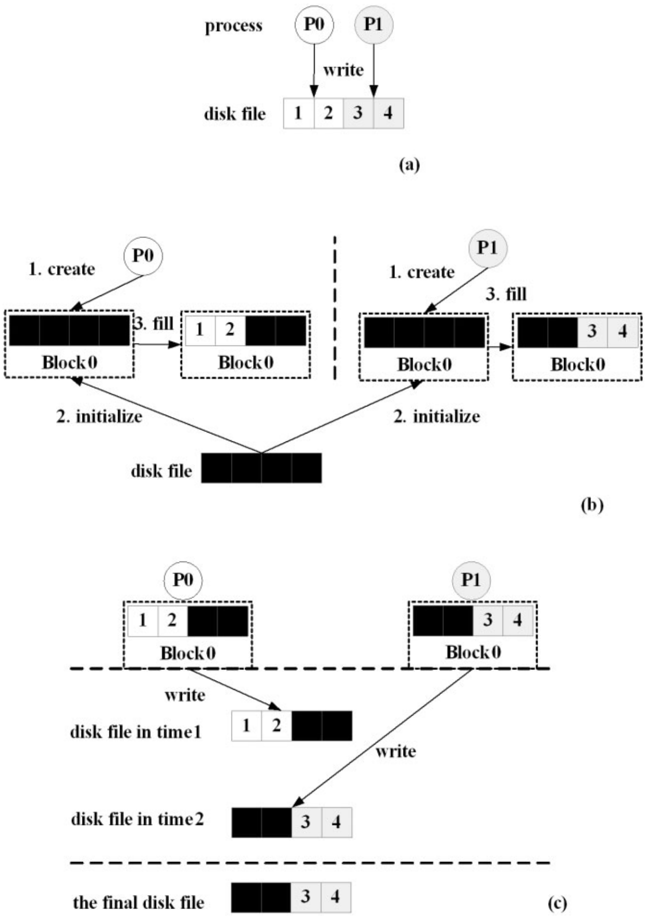

Figure 6 Analysis of the reason for the incorrect results produced by *GDAL_PIO* with column- and block-wise decomposition (black cell means no data): (a) two processes are writing to respective subdomains of a raster file with column-wise decomposition (the size of the GDAL cache block is 1 × 4, i.e. one row of the raster); (b) each process prepares data in the cache block; and (c) each process writes the cache block in memory to the same region of the shared disk file

is executed with column- or block-wise decomposition. This situation means that each call to the *RasterIO()* function will involve multiple I/O requests. For a large raster array, the large number of small I/O requests generated from column- or block-wise decomposition will considerably augment the I/O time (Figure 5c).

Incorrect results from *GDAL_PIO* with column- and block-wise decomposition are attributable to the caching mechanism in GDAL. GDAL maintains in memory the cache blocks fetched from the raster file and ensures that a second attempt to access the same block will be served from the cache instead of from the file. Such a caching mechanism makes the cache block to be the smallest size of each I/O request. The width of the cache block in GDAL is often equal to the raster width. In this situation, incorrect results will arise when different processes try to write the cache block to the same region of a shared raster file.

Consider a simple case in which two processes are writing to two subdomains of a raster file with column-wise decomposition (Figure 6a). In this case, the width and height of the GDAL cache block are four and one, respectively. Process *P0* first creates a cache block in memory (black cell means no data), initializes it from the disk file, and then sends the output value to the cache block (Figure 6b). Meanwhile, process *P1* performs the same tasks as *P0* (Figure 6b). After the output value has been sent to the cache block, each process writes the cache block in memory to the same region of the raster file on disk. As shown in Figure 6c, the file is first written by process *P0* at time 1 and later is rewritten by process *P1* at time 2. Therefore the final results file contains cells without data (Figure 6c), as shown by the experiment described earlier (Figure 4b). In fact, the results file will always contain some cells with no data unless the processes write the cache block serially to the same region of the file.

4 Improvement Based on a Two-Phase I/O Strategy

This section introduces a "two-phase I/O" strategy which is a suitable solution for the problems in *GDAL_PIO* with column-wise and block-wise decomposition. The two-phase I/O strategy was first proposed by Rosario et al. (1993) and was implemented in MPI I/O for a much more efficient use of the I/O subsystem (Thakur et al. 1999). The basic idea of this strategy is to redistribute data among the processes by combining a large number of small I/O requests into a small number of large continuous I/O requests.

Based on the two-phase I/O strategy, a data redistribution module for GDAL was designed. This module provides two features. The first is to reduce the number of I/O requests by combining small I/O requests into large continuous I/O requests through inter-process communication. This should be efficient because I/O operations are often more time-consuming than inter-process communication. The second feature is to avoid possible conflict among processes which might be trying to write the cache block to the same region of the shared raster file.

Figure 7 shows an example of a write operation involving two processes and using such a data redistribution module. The two processes first exchange their local individual I/O requests to form new I/O requests. Then according to the new I/O request, inter-process communication is used to redistribute the data so that each process has one contiguous chunk of data. This means that the data for each process are available to be written to the disk file with only one I/O request. Moreover, because these two processes write their respective cache blocks to different regions of the shared raster file, there is no conflict.

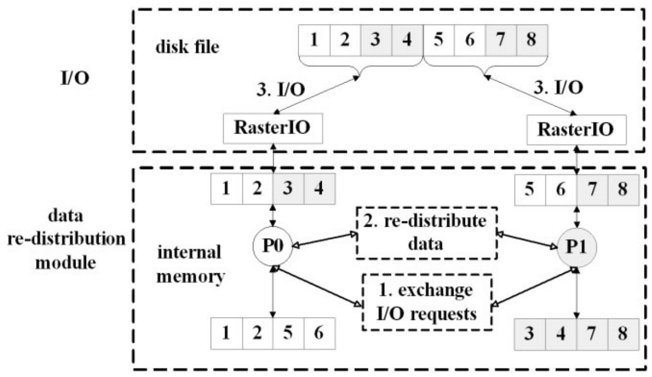

Figure 7 Using GDAL with a data redistribution module based on the two-phase I/O strategy

5 Experiment and Results

MPI was used to implement the proposed data redistribution module based on the two-phase I/O strategy for using GDAL in parallel raster I/O mode (called *GDAL_RDPIO*, short for "*ReD*istribution *P*arallel *I/O*"). The test data and test environment are the same as for the earlier experiment (see Section 3.1).

The experimental results show that *GDAL_RDPIO* was about four times faster than *GDAL_PIO* when column- or block-wise decomposition was used (Figure 4a). With row-wise decomposition, *GDAL_RDPIO* and *GDAL_PIO* achieved almost the same I/O times. The results from *GDAL_RDPIO* with each of the three data decompositions are correct. *GDAL_RDPIO* shows satisfactory efficiency and feasibility.

6 Conclusions

This article first explores the parallel raster I/O mode of using GDAL to access geospatial raster data. Experimental results show that parallel raster I/O using GDAL under column- or block-wise domain decomposition is highly inefficient and cannot achieve correct output. After analysis of the reason for the problems with parallel raster I/O using GDAL, a two-phase I/O strategy was used to design a data redistribution module for GDAL to fix these problems. MPI was used to implement this data redistribution module for GDAL. Experimental results show that the proposed data redistribution module provides an efficient solution to effective application of GDAL to parallel geospatial raster I/O.

References

Guan Q and Clarke K C 2010 A general-purpose parallel raster processing programming library test application using a geographic cellular automata model. *International Journal of Geographical Information Science* 24: 695–722

Maulik U and Sarkar A 2012 Efficient parallel algorithm for pixel classification in remote sensing imagery. *Geoinformatica* 16: 391–407

Qin C-Z and Zhan L-J 2012 Parallelizing flow-accumulation calculations on graphics processing units: From iterative DEM preprocessing algorithm to recursive multiple-flow-direction algorithm. *Computers and Geosciences* 43: 7–16

Rosario J M, Bordawekar R, and Choudhary A 1993 Improved parallel I/O via a two-phase run-time access strategy. *ACM SIGARCH Computer Architecture News* 21(5): 31–8

Shook E and Wang S-W 2011 A parallel input-output system for resolving spatial data challenges: an agent-based model case study. In: *Proceedings of the Second ACM SIGSPATIAL International Workshop on High Performance and Distributed Geographic Information Systems*, Chicago, Illinois: 18–25

Thakur R, Gropp W, and Lusk E 1999 On implementing MPI-IO portably and with high performance. In *Proceedings of the Sixth Workshop on I/O in Parallel and Distributed Systems*, Atlanta, Georgia: 23–32

Warmerdam F 2008 The geospatial data abstraction library. In Hall B and Leahy M G (eds) *Open Source Approaches in Spatial Data Handling*. Berlin, Springer: 87–104

Wang X, Li Z, and Gao S 2012 Parallel remote sensing image processing: taking image classification as an example. In Li Z, Li X, Liu Y, and Cai Z (eds) *Proceedings of the Sixth International Symposium on Intelligence Computation and Applications, Wuhan, China, 27–28 October, 2012*. Berlin, Springer-Verlag: 159–69