remote sensing

MDPI

Article

A Rapid Parallel Mosaicking Algorithm for Massive Remote Sensing Images Utilizing Read Filtering

Pei Nie [1,2], Zhenqi Cui [2,3,*] and Yaping Wan [1]

[1] College of Computer Science, University of South China, Hengyang 421001, China; p.nie@utwente.nl (P.N.); ypwan@usc.edu.cn (Y.W.)
[2] Faculty of Geo-Information Science and Earth Observation (ITC), University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands
[3] School of Geosciences and Info-Physics, University of Central South, Changsha 410012, China
* Correspondence: z.cui@utwente.nl

Abstract: Mosaicking is a crucial step in the application of remote sensing images. The amount of remote sensing image data has grown rapidly, along with the expansion of observed areas and increased image resolution. As a result, traditional serial mosaicking techniques are facing significant challenges. In recent times, various studies have utilized high-performance computing to hasten image mosaicking and attain favorable outcomes. Nevertheless, the current research only accelerates mosaicking through external technology, without optimizing from the perspective of algorithm flow, which introduces unnecessary data I/O and slows down the mosaicking. This paper introduces a rapid parallel remote sensing image mosaicking algorithm utilizing read filtering. To begin with, the target images are divided into blocks and stored in a distributed file system. Subsequently, the image blocks are read and filtered based on a designated input format. Finally, the overlapping and non-overlapping areas are read and processed asynchronously, reducing the data I/O and computing overhead, thereby improving the efficiency of parallel computing. The experiments indicate that the mosaicking algorithm introduced in this paper enhances throughput and speedup by an average of 1.38 MB/S and 0.87 relative to the current techniques, respectively, concerning various datasets and cores. This study provides a theoretical foundation and novel ideas for processing remote sensing images on cluster platforms.

Keywords: remote sensing image; image mosaicking; parallel processing; read filtering; Spark

check for updates

Citation: Nie, P.; Cui, Z.; Wan, Y. A Rapid Parallel Mosaicking Algorithm for Massive Remote Sensing Images Utilizing Read Filtering. *Remote Sens.* **2023**, *15*, 4863. https://doi.org/10.3390/rs15194863

Academic Editor: Amin Beiranvand Pour

Received: 17 August 2023
Revised: 29 September 2023
Accepted: 1 October 2023
Published: 7 October 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Image mosaicking is defined as the procedure of seamlessly stitching together images that share overlapping regions, creating a unified and coherent image (see Figure 1) [1]. Given the wide range of applications for large-scale remote sensing images in fields such as environmental monitoring, resource surveillance, and disaster risk assessment [2–5], image mosaicking serves as a vital component of the remote sensing image processing pipeline [6]. It plays a pivotal role in analyzing and visualizing extensive spatial information captured by remote sensing platforms, enabling the creation of detailed and seamless representations of landscapes. Consequently, it facilitates improved comprehension and decision-making in diverse domains, including agriculture, forestry, environmental monitoring, and urban development.

For successful mosaicking, three criteria must be met: (1) consistent geometric shapes of objects in the resulting image, (2) normalized radiation intensity, and (3) seamless transitions and natural connections in overlapping areas. To satisfy these criteria, remote sensing image mosaicking typically involves five steps: image registration, overlapping areas extraction, radiometric normalization, seamline detection, and image blending [7] (see Figure 2). Image registration [8] ensures criterion (1) by matching the same features in different images to keep their geometric shapes consistent. After image registration, extracting the overlapping areas serves as the foundation for subsequent processing. Depending

on whether the remote sensing image has spatial geographic reference information, the extraction of overlapping areas can be based on geographic reference [9] or pixel similarity [10,11]. If the images are obtained from varying sensors or in different time periods, the radiation intensity between them may vary, hence leading to an unnatural appearance in the mosaicked result. To preserve criterion (2), radiometric normalization [12] (also referred to as radiometric balancing or tonal adjustment) computes the radiation mapping relationships for the entire image using overlapping areas. Seamline detection [13] and image blending [14] ensure criterion (3). Seamline detection locates the best position for splicing the images, and based on the seamline, image blending reduces differences around the seamline and then integrates them into a cohesive whole.

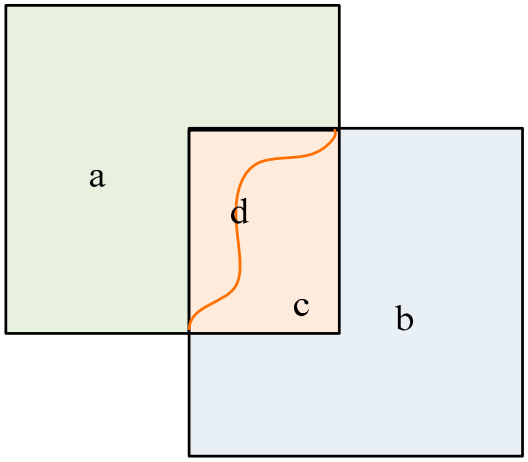


Figure 1. Image mosaicking; c presents the overlapping area of image a and image b; d is the seamline.

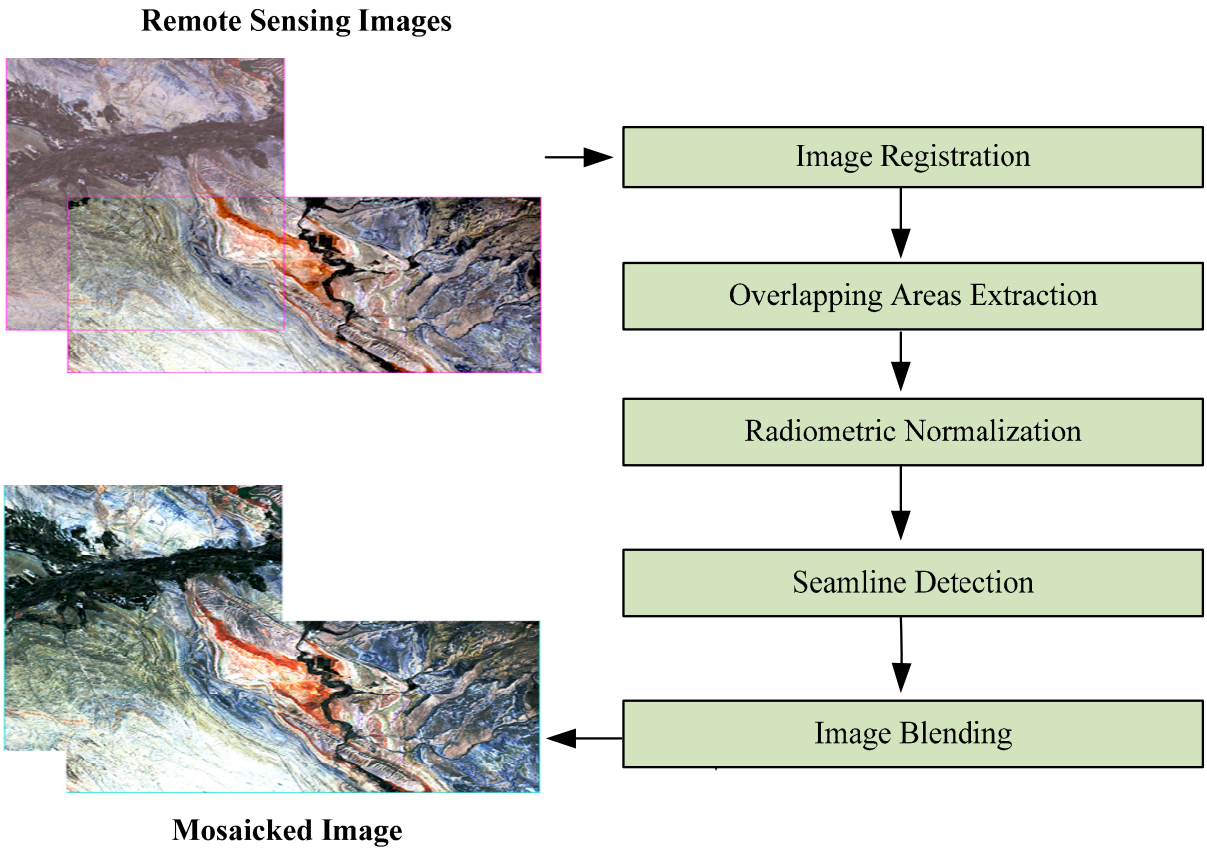


Figure 2. The process of remote sensing image mosaicking.

In some applications, such as disaster monitoring or time-sensitive analysis, there is a need for real-time or near-real-time processing of remote sensing data. However, we can see that remote sensing image mosaicking necessitates significant computing resources and data, especially when dealing with high-resolution or multispectral imagery, where large amounts of image data are loaded into the memory at one time, and data are transferred frequently between mosaicking steps, which brings huge data computing and input/output (I/O) load to a single computer. Thus, it is deemed useful to design parallel mosaicking algorithms based on high-performance computing (HPC). Presently, HPC solutions addressing the formidable task of mosaicking massive remote sensing images mainly encompass CUDA-programmed GPUs [15], parallel computing clusters with MPI (message passing interface) support [16], and Spark-based distributed computing [17]. Eken [18] introduced a novel concept centered on the assessment of available hardware resources within the host machine where the mosaicking process takes place. This evaluation serves as the basis for dynamically scaling the image resolution accordingly. Ma [15] delved into the development of an efficient, reusable, GPU-based model for processing remote sensing images in parallel. They established a collection of parallel programming templates designed to simplify and enhance the creation of parallel algorithms for remote sensing image processing. However, when dealing with vast volumes of remote sensing data, achieving optimal mosaic results becomes a complex challenge that cannot be solely addressed by enhancing the standalone performance of computing devices and coordinating logical operations in parallel. Furthermore, the use of CUDA programming, geared toward computationally intensive tasks, necessitates intricate configurations for various complex logic operations, thus elevating the programming complexity and limiting flexibility.

Parallel processing architectures, including centralized cluster structures and physically dispersed distributed structures, introduce new perspectives for the processing of massive remote sensing images. In Chen's study [19], three challenges in mosaicking parallelism were discussed: difficulty in handling multiple dependent tasks, multistep programming, and frequent I/O operations. To schedule mosaicking tasks that handle a vast number of dependent tasks, Wang [20] represented the priority of the task list with a minimal spanning tree. On the other hand, Ma [16] introduced a task-tree-based approach for dynamic directed acyclic graph (DAG) scheduling of massive remote sensing images rather than static task scheduling. The most common solutions for parallel mosaicking programming are based on MPI [21,22] and on MPI + OpenMP [16]. Although these programming paradigms enable parallel mosaicking, they require consideration of low-level parallel algorithm details, and the difficulty in multistep programming has not been solved. Moreover, frequent data loading and exporting operations in large-scale image mosaicking introduce significant data I/O overheads, which are parallelized mainly through multithreading [20] and distributed file systems [16,23]. While the aforementioned research aims to solve the difficulties in mosaicking parallelization, the implemented algorithms are often complicated. Compared to GPU and MPI, Apache Spark [24] is a high-level parallel computing framework that generates tasks into a DAG for scheduling, with the ability to directly interact with distributed file systems. It allows users to call the Spark API for parallel computing, without concern for the details of the underlying implementation. In light of this, Wu [25] designed a parallel drone image mosaicking method harnessing the power of Spark. By modifying it to be fit for fast and parallel running, all steps of the proposed mosaicking method can be executed in an efficient and parallel manner. For large-scale aerospace remote sensing images, Jing [17] proposed a parallel mosaicking algorithm based on Spark. With the data to be mosaicked stored in a distributed file system, parallel mosaicking is achieved through a custom Resilient Distributed Dataset (RDD), without requiring consideration of too many underlying parallel details. This improvement markedly enhances the efficiency of parallel mosaicking. Ma [26] introduced a large-scale, in-memory, Spark-enabled distributed image mosaicking approach. By utilizing Alluxio for data prefetching and expressing the data as RDDs for concurrent grid-based mosaicking tasks within a Spark-enabled cluster, this method minimizes data transfers and

enhances data locality. The experiments indicate that this approach significantly improves the efficiency and scalability of large-scale image mosaicking compared to traditional parallel implementations.

Although HPC, especially Spark, has significantly improved the performance of remote sensing image mosaicking, these studies have used external advanced technologies to empower mosaicking, and they have not deeply studied the characteristics of mosaicking or optimized it from the perspective of algorithm flow. As presented in Figure 1, the primary focus of image mosaicking processing is the overlap area. Operations such as overlapping area extraction, seamline detection, and image blending only deal with overlapping regions. On the other hand, for image registration and radiometric normalization, mapping relations are acquired from overlapping regions and applied to the entire image. For example, control points of mutual information are derived from overlapping regions before constructing the global geometric polynomial. However, the current parallel mosaicking lacks a designed image storage structure. All image data, both overlapping and non-overlapping, are loaded into memory for the process, resulting in the entire global image being resident in memory during the mosaicking. This undoubtedly increases the node's load and slows down the mosaicking process.

Addressing the issue commonly encountered in the existing parallel mosaicking techniques, this paper introduces each step of the mosaicking in detail, optimizes the mosaicking from the perspective of the algorithm flow, and proposes a rapid parallel mosaicking algorithm that utilizes read filtering to process the overlapping area and non-overlapping area asynchronously. The algorithm comprises three main components, namely, preprocessing, read filtering, and mosaicking processing: (1) For preprocessing, we first calculate the overlapping areas of the images and then divide the images into blocks and store them in a distributed file system. (2) For read filtering, image blocks within the overlapping areas are read from the distributed file system, and blocks outside the overlapping areas are filtered out to reduce the I/O load. (3) For mosaicking processing, after read-filtering, only the overlapping areas are loaded into memory for mosaicking processing. Data outside the overlapping areas undergo image registration and radiometric normalization based on the mapping relationship determined from the overlapping areas. Once the mosaicking results of the overlapping and non-overlapping areas are obtained, the results are combined to generate a complete mosaicking image. Our algorithm is designed to utilize read filtering to effectively minimize the I/O load, improving the overall efficiency and processing speed. The proposed approach efficiently addresses common issues in parallel mosaicking techniques, making it a promising solution for the mosaicking of large-scale remote sensing images.

2. Materials and Methods

2.1. Principle of Remote Sensing Image Mosaicking

2.1.1. Image Registration

Image registration ensures that the geometric shape and spatial position of an object in images captured at different times and from different viewpoints are consistent. There are two main categories of existing techniques: area-based and feature-based methods [27]. Area-based methods use pixel values, whereas feature-based methods use low-level features of the image. Among various image registration methods, mutual information is known for its high accuracy and ease of implementation [28]. As a result, we selected the mutual information method to perform the registration of images.

In mutual-information-based image registration, a reference image and a second image are required for the registration process. The first step involves extracting control points by analyzing the mutual information between the images. The purpose of this step is to identify the area with the highest mutual information. Next, a geometric polynomial is constructed based on these control points to create a mapping function between the row–column coordinates and the geographic coordinates. The geometric polynomial serves as the mapping function used to establish the relationship between the two sets of coordinates.

Finally, the registered image is generated through resampling, which allows the pixel values of the second image to be transformed and aligned with those of the reference image in a consistent manner. By following these steps, the mutual-information-based registration approach can effectively ensure that the geometric shape and spatial position of objects in images captured at different times and viewpoints remain consistent.

Geometric polynomials and resampling are widely used techniques in image registration. Thus, this paper focuses solely on the calculation of images' mutual information. Mutual information is a measure that describes the amount of information shared between two systems or data sources.

In image registration, mutual information is quantified by analyzing entropy and joint entropy. Entropy H is calculated using Formula (1), where h_i represents the total number of pixels with gray level i in the image, p_i is the probability that a pixel has gray level i, and N is the total number of gray levels in the image.

$$H = -\sum_{i=0}^{N-1} p_i \log p_i \qquad p_i = h_i / \sum_{i=0}^{N-1} h_i \tag{1}$$

The formula for computing the joint entropy $H(X,Y)$ of images X and Y is shown in Formula (2). The joint probability $P_{ij}(i,j)$ of the two images can be calculated using a joint histogram [29], i represents the pixel gray level in image X, and j represents the pixel gray level in image Y.

$$H(X,Y) = -\sum_{i,j} P_{ij}(i,j) \log P_{ij}(i,j) \tag{2}$$

The mutual information $MI(X,Y)$ can be calculated using Formula (3) from the entropy and joint entropy of images X and Y.

$$MI(X,Y) = H(X) + H(Y) - H(X,Y) \tag{3}$$

2.1.2. Overlapping Areas Extraction

Extracting overlapping areas is a simple process. When the remote sensing images are georeferenced, the georeferencing information, such as geographical coordinates (e.g., GPS data) or attitude data and onboard position (e.g., inertial navigation system data), can be used directly to extract these areas. For images without georeferencing information, other methods, such as phase correlation [10] or scale-invariant feature transform (SIFT) [11], can be used to calculate the pixel similarity between images for identifying overlapping areas.

2.1.3. Radiometric Normalization

Remote sensing images, often captured at different times or using different sensors, exhibit significant radiation variations. These variations lead to visually inconsistent images after mosaicking. To ensure tonal balance and visual consistency across the mosaic, radiometric normalization is imperative.

Radiometric normalization operates under the assumption that the reflection conditions in the overlapping regions of two images remain static. As such, pixel pairs from the overlapping areas of two images are used to compute the mapping relationship. This relationship is then extended to the global image, as depicted in Figure 3. Radiometric normalization has three primary methods: global models, local models, and combined models [7]. Global models are the most widely adopted method, establishing a linear or nonlinear mapping function based on overlapping pixel pairs. One image serves as a reference (source image), while the other is normalized (target image). The global model is mathematically represented as shown in Formula (4):

$$M_1^* = f(M_1) \tag{4}$$

where M_1 denotes the target image, with M_1^* referring to the normalized image, and $f(\ )$ representing the linear or nonlinear function that governs the global image mapping

relationship. The mapping function can be derived in diverse ways, including linear regression [30] and least-mean-square (LMS)-based transformation [31].

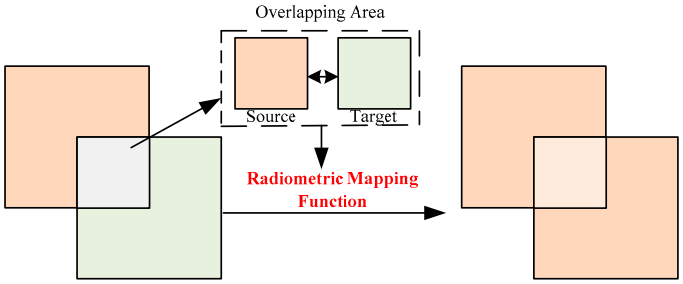


Figure 3. Radiometric normalization.

2.1.4. Seamline Detection

Detecting the optimal seamline is a critical step in creating seamless image mosaics. The ideal seamline is characterized by the closest match in pixel values and texture features in the overlapping areas of the two images. Typically, the seamline of the overlapping area is a curved line (see Figure 1), and the highest-quality output is achieved by cutting and splicing the images along this line.

The two primary methods for detecting seamlines are image-internal-information-based and external-data-based [7]. Since the former is more commonly used, we will focus on this method. The image-internal-information-based method utilizes information within the image, such as pixel values and texture structure, to detect the optimal seamline. This method can be categorized into two detection processes: frame-to-frame and multiframe joint methods (as depicted in Figure 4). With frame-to-frame detection, seamlines are detected individually, whereas multiframe detection can identify multiple seamlines concurrently. Frame-to-frame detection is more suitable for efficient parallel processing and can be achieved by implementing multiple nodes. The primary frame-to-frame methods include the bottleneck model [32], the snake model [33], Dijkstra's algorithm [34], and the DP algorithm [35]. The simplest and most direct method is the bottleneck model, which uses pixel value differences between images to define the cost function, as shown in Formula (5):

$$C_{ij} = \left| L_{ij} - R_{ij} \right| \tag{5}$$

where the value of pixel (i, j) is L_{ij} for one image and R_{ij} for the other, while the cost C_{ij} is defined as the absolute difference between them. If the overlapping area contains a total of M*N pixels, then any seamline (SL) is a straightforward path from a row 1 pixel to a row M pixel. Equation (6) represents the cost of any SL:

$$C(SL) = \sum C_{ij}, (i, j) \in SL \tag{6}$$

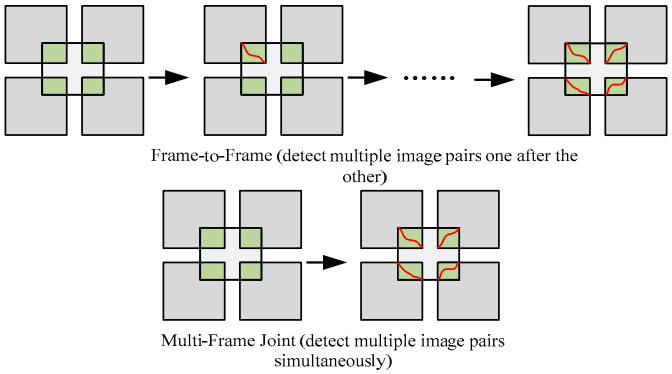


Figure 4. Frame-to-frame and multiframe methods; seamlines are highlighted in red.

The problem of identifying the optimal seamline is equivalent to minimizing $C(SL)$.

2.1.5. Image Blending

Although the optimal seamline displays the path with the least difference, there could be radiometric variation around the seamline. Therefore, image blending is necessary to eradicate inconsistency around the seamline and achieve a seamless transition between the images. As a result of the blending process, a complete mosaic result image is obtained.

The weight combination of transition zones is the most frequently applied blending method [7]. This combination is realized through a weight function, typically using cosine distance-weighted blending (CDWB) [6]. As illustrated in Figure 5, CDWB creates a buffer along the seamline, known as the transition zone. The pixels within the transition zone result from the weighted combination of the two images, while pixels outside the buffer are obtained from the left and right images on the respective sides of the overlap region. CDWB uses distance as the weight criterion. Thus, pixels closer to the image have a greater weight. Figure 5 displays that for a pixel P(i, j) in the buffer, d is half of the buffer length, while d-di represents the distance from the pixel to the left buffer border. The weight of the left image increases as P(i, j) gets closer to the left buffer border, and vice versa for the right image. The expression of the distance ratio S of P(i, j) to the seamline is provided in Formula (7):

$$S = \frac{d - d_i}{2d}, -d \le d_i \le d \tag{7}$$

where the distance d_i from P(i, j) to the seamline is negative when P(i, j) is on the left side of the seamline, while it is positive when P(i, j) is on the right side. The complete mosaicked image, I, can be obtained using Formula (8):

$$I(i,j) = \begin{cases} L(i,j), \ (i,j) \in L \\ W_L(S)L(i,j) + W_R(S)R(i,j), (i,j) \in (L \cap R) \\ R(i,j), (i,j) \in R \end{cases} \tag{8}$$

where $W_L(S)$ and $W_R(S)$ are the weights of image L and image R in the buffer, respectively. These weights are functions of the distance ratio S, and they satisfy the conditions that $W_L(S)$ + $W_R(S)$ = 1 and 0 <= $W_L(S)$, $W_R(S)$ <= 1. This relationship is depicted in Formula (9):

$$\begin{cases} W_L(S) = -\frac{1}{2}\cos(\pi S) + \frac{1}{2} \\ W_R(S) = 1 - W_L(S) = \frac{1}{2}\cos(\pi S) + \frac{1}{2} \end{cases} \tag{9}$$

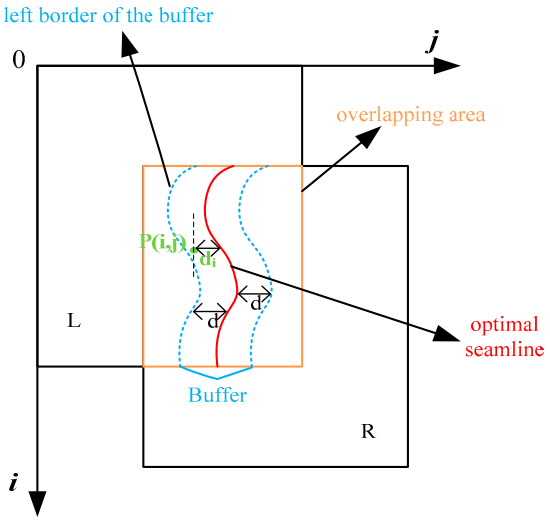


Figure 5. Image blending using CDWB.

2.2. Overview of the Proposed Algorithm

The core of mosaicking processing is the overlapping area. In fact, overlapping area and non-overlapping area can be processed separately. In this paper, we propose a rapid parallel mosaicking algorithm based on read filtering, as illustrated in Figure 6. The algorithm consists of three parts: preprocessing, read filtering, and mosaicking processing. In the preprocessing phase, we first extract the overlapping area of the images, divide the images into blocks, and store the blocks in the Hadoop Distributed File System (HDFS). Read filtering is based on the overlapping area and involves reading the intersecting data blocks from the HDFS, filtering out data that fall outside the overlapping area, and reducing the I/O load. Finally, we use Spark to perform parallel mosaicking processing. After the read-filtering process, the overlapping area is read into the distributed memory to form an RDD. For data outside the overlapping area, we load them into memory after the mapping function is obtained from the overlapping area, and then we perform image registration and radiometric normalization. After obtaining the mosaicking results of the overlapping and non-overlapping areas, we combine the two areas to form a whole mosaicking result image.

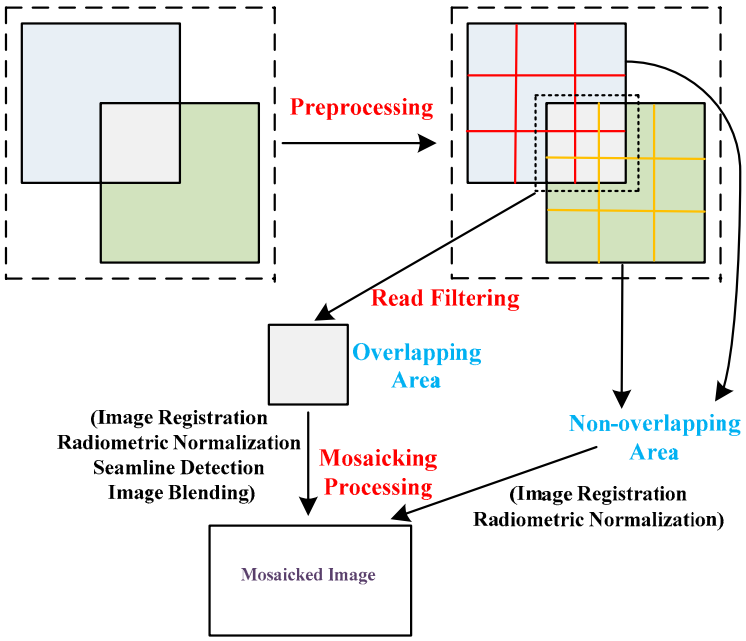


Figure 6. Parallel mosaicking algorithm framework.

2.3. Preprocessing

To distinguish between the overlapping and non-overlapping areas in the mosaicking process, we introduce preprocessing. First, we extract the overlapping area, and then we divide the images into blocks, which are stored in the Hadoop Distributed File System (HDFS). The intersecting relationship between the overlapping area and the image blocks forms the basis for read filtering.

To accurately extract the overlapping area, we utilize georeferencing information (if available) and scale-invariant feature transform (SIFT) to extract the union of the overlapping areas. Once the overlapping area is extracted, we divide the images into fixed rectangular blocks and assign a number to each block. Figure 7 shows that two images, L and R, are partitioned into 9 rectangles, each labeled with a row and column number. The overlapping area intersects with blocks (2,2), (2,3), (3,2), and (3,3) of image L and blocks (1,1), (1,2), (2,1), and (2,2) of image R, which constitutes the mapping relationship between the overlapping area and the image blocks: Map (overlapping area) = {L (2,2), L (2,3), L (3,2), L (3,3), R (1,1), R (1,2), R (2,1), R (2,2)}.

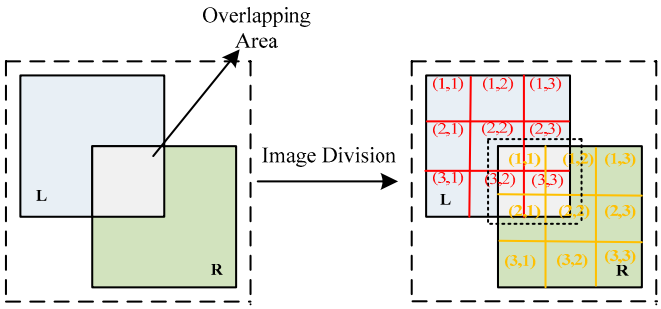


Figure 7. Image division.

Distributed storage provides underlying data support for parallel processing, and concurrent computing nodes can read data from distributed storage. The open-source big data system, Hadoop Ecosystem, offers a two-tiered framework of distributed storage (HDFS) and parallel processing using MapReduce and Spark. HDFS is a fault-tolerant file system suitable for deployment on commodity hardware with efficient data access. In this paper, we design a storage structure for images using HDFS's MapFile [36].

As shown in Figure 8, MapFile consists of two key–value files: one for storing data, and the other for storing indices. The data in MapFile are sorted by keys, and an index is generated using the key and the offset of each key value in the data file. Image blocks that are partitioned are stored in the data file of MapFile. Each image block corresponds to a key–value pair, in which the key denotes the image block number, while the value represents the serialized image block stream. For the image block number (key) and the serialized image block stream (value), we store them using two bytes and a byte array, respectively. The key occupies two bytes, and the correspondence between the image block number (i.e., row number and column number) and the key is shown in Figure 9. The row occupies the first byte, and the column occupies the next byte. With a maximum of 27 rows and columns, there are a total of 27^2 image blocks. The maximum capacity of the block stream is determined by setting the array length.

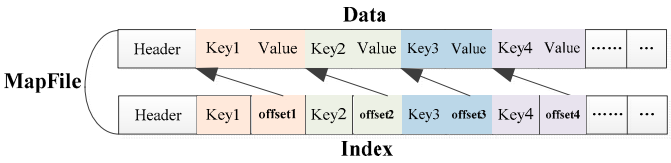


Figure 8. MapFile.

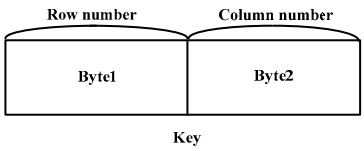


Figure 9. The correspondence between the block number and key.

2.4. Read Filtering

Preprocessing describes the storage of images in the MapFile in blocks, as well as their retrieval based on the mapping relationship with the overlapping area. To achieve asynchronous reading of both overlapping and non-overlapping areas, a special input format of MapFile called read filtering is utilized. MapReduce/Spark applications are divided into multiple subtasks, with each subtask processing a particular input data split that is further subdivided into records. A record corresponds to a key–value pair, and the split data are read in parallel at each task node. The input data split represents a logical concept that corresponds to a physical HDFS data block. In Hadoop, this split is abstracted into the InputSplit class, which contains information such as the split length

field in bytes and a method to obtain the data location. The interaction between the computing framework and the HDFS is shown in Figure 10. It can be seen that InputFormat is the interface for the parallel computing framework to read from the HDFS. Based on InputFormat, the mapping between HDFS data blocks and data splits of the task is established.

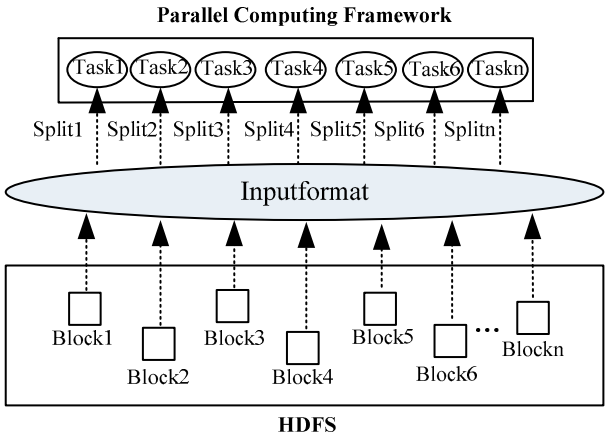


Figure 10. The interaction between the parallel computing framework and the HDFS.

Creation and segmentation of the input data split is handled by the abstract class InputFormat and its subclasses. As the parent class of all InputFormats, the abstract class InputFormat consists of two abstract methods:

Public abstract list<InputSplit> getSplits (JobContext context):

This method is utilized to acquire the input split, which is then forwarded to the master node. The master node schedules tasks based on the storage location information and assigns input splits in the closest possible proximity to each task.

Public abstract RecordReader<K,V> createRecordReader (Input Split split, TaskAttemptContext context):

This method creates an iterator object called RecordReader to traverse through the specified input split. The RecordReader is then utilized to divide the input split into records (key–value pairs) that can be processed by the task.

The above statement reveals that InputFormat specifies rules that the parallel computing framework must abide by while reading the HDFS. The native InputFormat for MapFile is SequenceFileInputFormat, which extends FileInputFormat. The SequenceFileInputFormat class provides an implemented createRecordReader method. This method is responsible for creating a data iterator for the input split. The iterator creates a reader instance, identifies the input split's location within the data, and iterates over records from beginning to end. Each record is then passed on to the task for processing. This approach allows for efficient reading of data in MapFile and easy integration with the parallel computing framework. Reading MapFile based on SequenceFileInputFormat is shown in Figure 11.

However, when reading the MapFile based on SequenceFileInputFormat, the index is not utilized. Instead, all key–value pairs are read sequentially without regard for their location, which does not meet our specific requirements. Based on the mapping relationship between overlapping areas and image blocks, we can redefine the InputFormat of the MapFile to enable concurrent random reading of image data blocks and on-demand data retrieval. Thus, asynchronous reading of both overlapping and non-overlapping areas is achieved. In this section, we propose and implement MapFileInputFormat: a dedicated parallel InputFormat designed for MapFile. This InputFormat extends SequenceFileInputFormat, rewrites the createRecordReader method, and returns a specific split iterator. Its createRecordReader is as follows:

Public RecordReader<K,V> createRecordReader(Short[] args, InputSplit split, TaskAttemptContext context).

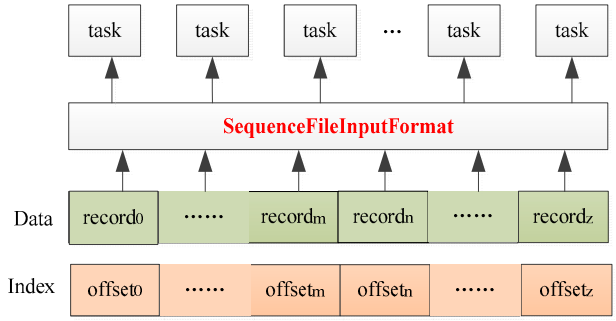


Figure 11. Reading MapFile based on SequenceFileInputFormat.

This method creates a MapFile iterator for the input split. For the split iterator, the split is read according to the method parameter args. The key–value records corresponding to the parameter args are read out and passed to the task. The key points of the split iterator are to return to the upper-level directory of the data file, create an instance of MapFile.Reader, load the index file into memory, and randomly read the input split according to the parameter args and index. In fact, when reading the overlapping area data, the method parameter Short[] args is the set of keys of the image blocks that intersect with the overlapping area.

The MapFileInputFormat is more flexible for parallel reading of MapFile. As illustrated in Figure 12, the MapFileInputFormat can randomly read image blocks according to the specified read parameter and index.

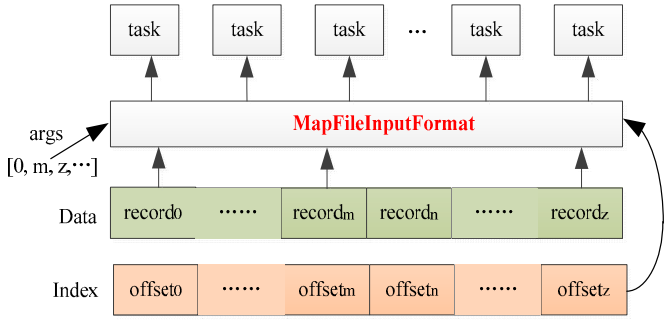


Figure 12. Reading MapFile based on MapFileInputFormat.

2.5. Mosaicking Processing

In this section, we employ Spark to accomplish parallel mosaicking processing of remote sensing images. Spark was proposed by the AMP Lab of Berkeley University in 2009 [24]. In summary, Spark is a comprehensive analysis engine for large-scale data processing that operates on clusters and delivers potent parallel computing capabilities. Resilient Distributed Datasets (RDDs) are the foundation of Spark, serving as a representation of distributed memory that is partitioned, read-only, and supports a broad range of operations. Operations on RDDs are categorized into input, transformation, and action.

During preprocessing, the images intended for mosaicking are separated into image blocks and saved in MapFiles. Spark randomly reads the MapFiles in parallel based on the MapFileInputFormat. The mapping relationship between the overlapping area and the image blocks enables asynchronous reading of the overlapping and non-overlapping areas. Firstly, the image blocks that intersect with the overlapping area are read into memory to generate an overlapping area RDD. Then, image registration, radiometric normalization, seamless detection, and image blending are executed. Next, the remaining image blocks are read into memory to form the non-overlapping area RDD, and image registration and radiometric normalization are carried out based on the mapping relationship computed from the overlapping area. Lastly, the overlapping area RDD and non-overlapping area RDD are merged to create the final image. In summary, Algorithm 1 illustrates a rapid parallel mosaicking algorithm for remote sensing images that leverages read filtering.

Algorithm 1. Rapid parallel mosaicking algorithm Spark-RF

Input: Images

1. OverlappingAreaExtraction(Images) → Array[OverlappingArea]
//Extract overlapping areas (OAs) between images and store them in an array; each OA is represented in either geospatial coordinates or row and column coordinates, depending on whether the image is georeferenced or not. The array functions as a shared Spark variable, allowing computing nodes to access it.
2. Preprocessing(Images) → MapFiles
//Divide the images into blocks and store to the HDFS in MapFiles.
3. For MapFile in MapFiles
3.1. Mapping(Array[OverlappingArea]) → Array[BlockIntersectOAs], Array[BlockOutsideOAs]
//Based on the relationship between each image and the OAs, an array of image blocks intersecting the OAs and an array of image blocks outside the OAs are obtained. Each image block in the array is identified by a block number.
3.2. Spark.ReadMapFile(MapFileInputFormat(Array[BlockIntersectOAs])) → OAPairRDD
//Based on the proposed MapFileInputFormat and block number, the blocks that intersect the OAs are read into memory to form OAPairRDD.
3.3. Spark.ReadMapFile(MapFileInputFormat(Array[BlockOutsideOAs])) → NonOAPairRDD
//Based on the proposed MapFileInputFormat and block number, the blocks outside the OAs are read into memory to form NonOAPairRDD.
4. For eachOAPairRDD
4.1. OAPairRDD.FlatMaptoPair(Array[OverlappingArea]) → FlatPairRDD
//By assessing the intersection between each image block and the OAs, we can determine the OAs that each image block covers. If a single block extends over multiple OAs, it must be assigned to all of those OAs. The block's key is then associated with the intersecting OA, and a field recording the image block number is added to the value of the block.
5. FlatPairRDD1.cogroup(FlatPairRDD2)...cogroup(FlatPairRDDn) → GroupPairRDD
//Given the preceding step's transformation, the FlatPairRDD key becomes indicative of the OA. To aggregate image blocks that share the same key, which signifies the same OA, we employ a cogroup operation.
6. GroupPairRDD.MapValues(MosackingProcessing) → MosaickedOAPairRDD
//Currently, the image blocks that intersect with same OA are collected together as one. Based on the block number in the value of the block (step 4.1), the blocks from the same image are combined in memory to form the mosaic area. For each mosaic area, mosaicking processing is performed based on the algorithms discussed in Section 2. It should be noted that the image registration and radiometric normalization mapping functions must be saved to the disk for the processing of image blocks outside the OAs in the next step.
7. NonOAPairRDD.MapValues(ImageRegistration and RadiometricNormalization) → MosaickedNonOAPairRDD
//Based on the registration and normalization mapping functions derived from the previous step, mosaicking of data outside the OAs is performed on the NonOAPairRDD derived from step 3.
8. MosaickedOAPairRDD.CollectAsMap( ) → OAMap
MosaickedNonOAPairRDD.CollectAsMap( ) → NonOAMap
//Return the mosaicking-processed OAPairRDD and NonOAPairRDD to the master node in the form of a map table.
9. Splice(OAMap,NonOAMap) → MosaickedImage
//On the master node, the map tables of OA and NonOA are merged to create the final complete image.
Output: Mosaicked Image

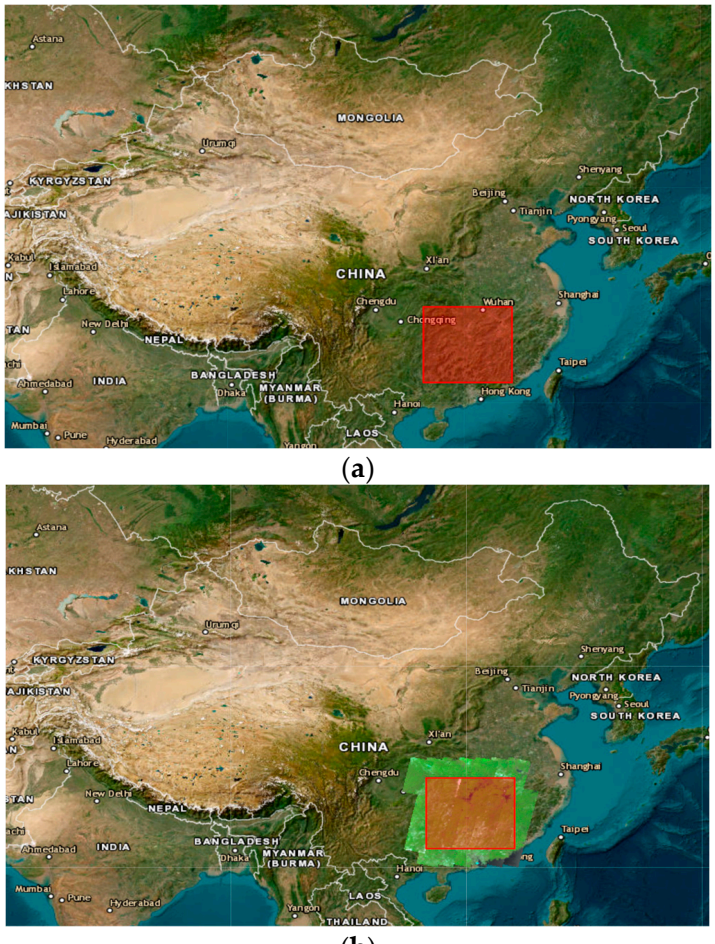
The flatMapToPair, cogroup, mapValues, and collectAsMap Spark operators are used in the parallel mosaicking algorithm described in Algorithm 1. The first three are transformation operations, while collectAsMap is an action operation. Note that in step 6, the mutual-information-based method is used for image registration, the LMS-based method

is used for radiometric normalization, the bottleneck-model-based method is used for seamline detection, and the CDWB algorithm is used for image blending.

3. Experimental Section

3.1. Study Area and Dataset

The study area selected for this paper is shown in Figure 13a. This area is located in South China, spanning Hunan, Guangdong, Hubei, Jiangxi, and Guangxi Provinces, ranging from (23.8557N, 108.6646E) to (30.9399N, 117.0801E). The experimental dataset consists of 48 images taken by the Landsat 8 satellite, using two sensors: the OLI and TIRS. The images come from the Landsat Collection2 Level-1 dataset; the path number is from 120 to 127, the row number is from 38 to 44, the total data volume is 55.28 GB, and the CRS is from WGS 84/UTM ZONE 49N to WGS 84/UTM ZONE 50N. Each image has 11 bands, among which bands 1–9 are collected by the OLI; band 8 (panchromatic) has a spatial resolution of 15 m, and the remaining eight bands are 30 m, while thermal infrared bands 10–11 are collected by the TIRS with a spatial resolution of 100 m. In order to reduce the noise of cloud cover in the mosaic, images from 2014–2015 and 2017–2022 with less than 30% cloud cover in July were selected. The overlap of the dataset and study area is shown in Figure 13b, and the 48 images in the figure constitute the Landsat natural color overview map.



(**a**)

(**b**)

Figure 13. Study area and dataset: (**a**) study area; (**b**) the overlap of dataset and study area.

3.2. Experimental Hardware and Software

For this experiment, a clustered environment with a master–slave architecture was employed, utilizing five Inspur I8000 blade servers equipped with Xeon E5-2620 v2 6-core

2.10 GHz processors, 32 GB of RAM, and 200 GB of hard disk storage. Each server had the necessary software suite, including Gcc 4.4.7, GDAL-2.3.1, Hadoop-2.5.2, Spark-1.5.0, and Openmpi-1.8.8, to enable the experiment. Spark was set to operate in standalone mode, the number of cores (SPARK_WORKER_CORES) allowed to be used by the application was the total number of cores of the node, and the memory (SPARK_WORKER_MEMORY) allowed to be used by the application was 16 GB. During the preprocessing stage, the image block size was set to $256 * 256$. The Spark application's submission parameters should be set as follows: set the executor memory to 2 GB, and the total executor cores to 24 cores. In addition, we propose the existing Spark and MPI parallel mosaicking for comparison purposes. The proposed mosaicking algorithm in this paper is identified as Spark-RF, and the comparison algorithm is referred to as Spark and MPI.

3.3. Mosaic Result Image

We show the fourth-band mosaic results of Spark-RF in Figure 14. Since the dataset spans two UTM projection zones, it is necessary to unify the 48 images into a unified coordinate system (WGS84, EPSG:4326) before performing the mosaic experiment. The images to be mosaicked are first divided into rectangular blocks and stored in MapFiles (preprocessing). Next, overlapping and non-overlapping regions are read asynchronously based on a MapFileInputFormat (read filtering), followed by the mosaicking processing. As can be seen from Figure 14, the view is continuous and there is no obvious stitching in the overlapping areas. To further quantitatively evaluate the image quality of parallel mosaicking, we selected the mosaicking result image of ENVI as a reference and evaluated the parallel mosaicking using two indicators: root-mean-square error (RMSE), and structural similarity index (SSIM). Table 1 presents the results, where a smaller RMSE value indicates a closer alignment between the two images, while a higher SSIM score (with a maximum value of 1) signifies greater structural similarity between the two images.

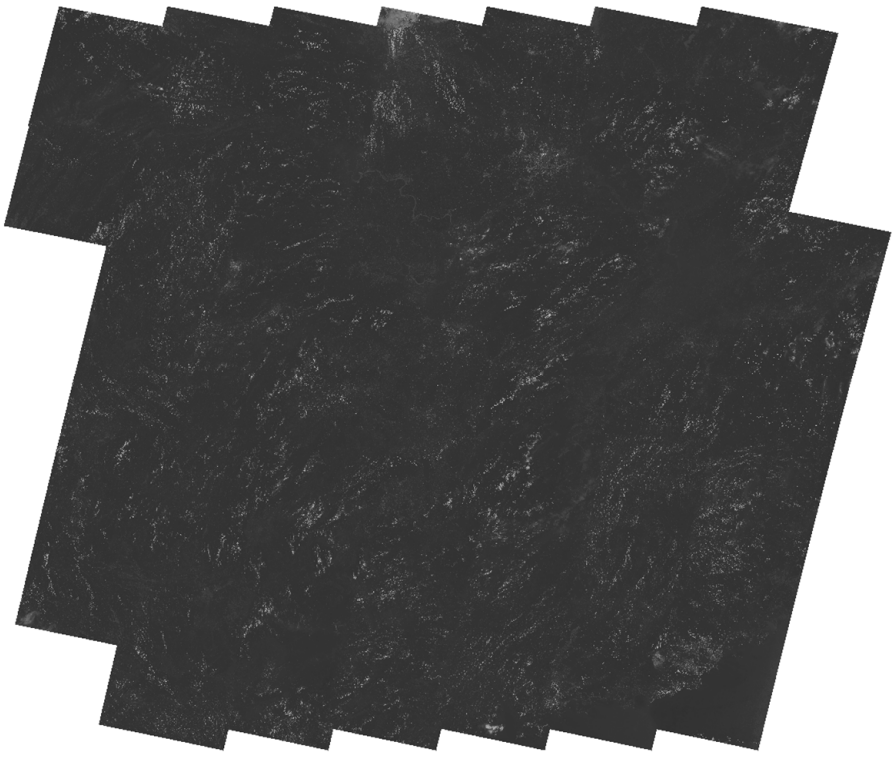


Figure 14. Mosaic result image.

Table 1. Quantitative evaluation results.

	Spark-RF	
	RMSE	SSIM
ENVI	81.3331	0.9989

3.4. Experiment 1: Efficiency

Experiment 1 compared the running time and throughput of Spark-RF proposed in this paper with Spark and MPI under different data volumes. From path 120 to 127, the data volume of the mosaic images increased in turn: 5 single-band images of path 120, 0.48 GB; 5 full-band images of path 120, 5.62 GB; 12 full-band images from path 120 to 121, 13.68 GB; 19 full-band images from path 120 to 122, 21.91 GB; 26 full-band images from path 120 to 123, 29.94 GB; 33 full-band images from path 120 to 123, 37.98 GB; 40 full-band images from path 120 to 125, 46.06 GB; 48 full-band images from path 120 to 127, 55.28 GB. Figure 15 and Table 2 show the running time of the algorithms. Figure 16 and Table 3 show the throughput. Throughput is calculated using Formula (10):

$$P = \frac{M}{T} \tag{10}$$

where P is the throughput, M is the data volume, and T is the running time of the mosaicking algorithm.

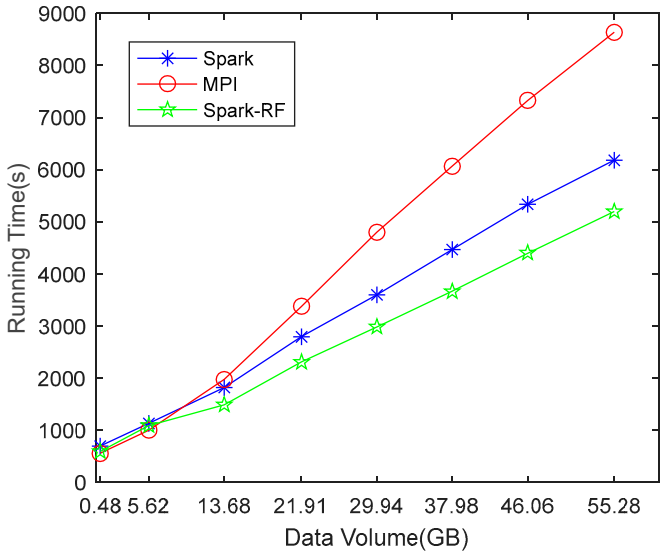


Figure 15. Running time of parallel mosaicking.

Table 2. Running time of parallel mosaicking.

	Running Time under Different Data Volumes							
Algorithm	**0.48 G**	**5.62 G**	**13.68 G**	**21.91 G**	**29.94 G**	**37.98 G**	**46.06 G**	**55.28 G**
MPI	561	1002	1976	3376	4900	6067	7329	8639
Spark	698	1133	1821	2797	3602	4466	5337	6186
Spark-RF	596	1089	1498	2312	2980	3669	4398	5199

It can be seen from Figures 15 and 16 that when the amount of data is small, MPI has less running time and higher throughput than the Spark-based mosaicking algorithm; as the amount of data gradually increases (greater than or equal to 13.68 GB), Spark and Spark-RF show higher efficiency and gradually widen the gap with MPI. This is because we use all of the cores of the node (24 cores in total) for the Spark cluster configuration. When

the amount of data is small, the number of concurrent tasks is greater than the number of tasks that need to be executed concurrently, resulting in some tasks being idle, and the communication between multiple concurrent tasks also slows down the mosaicking. When the amount of data increases, the number of tasks that the Spark cluster needs to execute concurrently increases, and the high concurrency of Spark is highlighted, which greatly improves the mosaic efficiency compared with MPI. We can also see that Spark-RF performs better than Spark on different datasets, because Spark-RF filters unnecessary data in the mosaicking and only involves necessary data in each mosaicking step, reducing the data I/O and computational load.

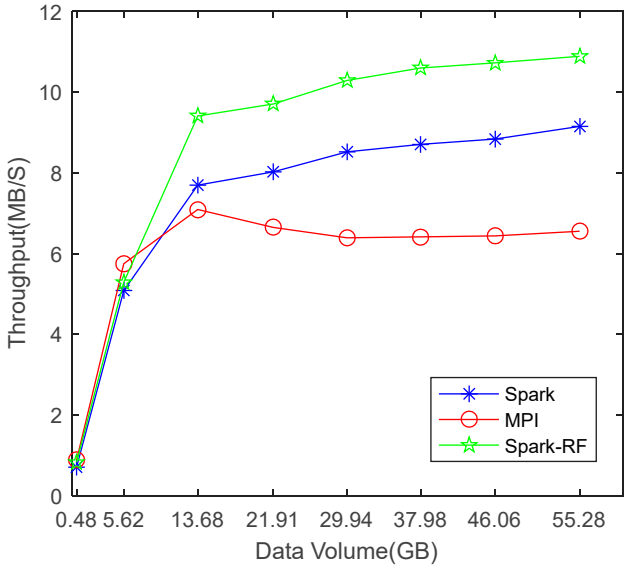


Figure 16. Throughput of parallel mosaicking.

Table 3. Throughput of parallel mosaicking.

	Throughput under Different Data Volumes (MB/s)							
Algorithm	**0.48 G**	**5.62 G**	**13.68 G**	**21.91 G**	**29.94 G**	**37.98 G**	**46.06 G**	**55.28 G**
MPI	0.88	5.74	7.09	6.65	6.39	6.41	6.44	6.55
Spark	0.70	5.08	7.69	8.02	8.52	8.71	8.84	9.15
Spark-RF	0.82	5.28	9.41	9.70	10.29	10.60	10.72	10.89

3.5. Experiment 2: Speedup

In Experiment 2, 48 full-band images with a total of 55.28 GB were selected. The main purpose of this experiment was to compare the speedup of each parallel mosaicking algorithm under different cores (processors) and then evaluate the scalability of the algorithms. We recorded the running time of the single-core mosaicking and parallel mosaicking algorithms under different numbers of cores. The speedup was calculated using Formula (11):

$$Speedup = \frac{T_s}{T_p} \tag{11}$$

where T_s represents the running time under a single core, while T_p represents the running time of parallel mosaicking with p cores. The speedup results are presented in Figure 17 and Table 4, where we can see that Spark-based parallel mosaicking has a higher speedup than MPI under different cores; this is because Spark caches the intermediate results in memory as RDDs, and MPI needs to frequently perform I/O processing on the intermediate results of each mosaicking step. As a result, MPI spends more time processing intermediate results. Furthermore, compared to the traditional Spark algorithm under the same dataset and cores, Spark-RF achieves higher speedup, and the advantage tends to increase with

the increase in the number of cores, which shows that Spark-RF has better parallelism. Finally, we can see that as the number of cores increases, the speedup of Spark-RF increases. When the number of cores is 4, the algorithm has the best scalability (2.53/4), and when the number of cores is 12, it has the worst (3.98/12). Overall, Spark-RF has better scalability than the other two algorithms. However, Spark-RF still has a long way to go from linear speedup, which has excellent scalability, because there is still a lot of node communication and coordination overhead in the Spark cluster.

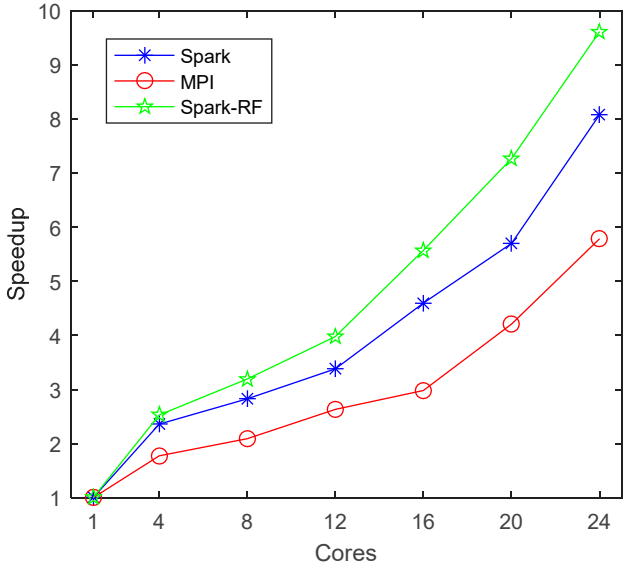


Figure 17. Speedup under different numbers of cores.

Table 4. Speedup under different numbers of cores.

	Speedup under Different Cores						
Algorithm	**1 Core**	**4 Cores**	**8 Cores**	**12 Cores**	**16 Cores**	**20 Cores**	**24 Cores**
MPI	1.00	1.77	2.09	2.63	2.98	4.21	5.79
Spark	1.00	2.36	2.83	3.38	4.60	5.70	8.08
Spark-RF	1.00	2.53	3.19	3.98	5.57	7.27	9.62

4. Discussion

The current parallel mosaicking algorithm mainly uses the characteristics of the parallel computing framework itself to parallelize the mosaicking task. MPI is based on multithreading and distributed file systems, while Spark is based on RDDs. However, remote sensing image mosaicking is not only a data-intensive task but also a computationally intensive task. If we can analyze the characteristics of the mosaicking algorithm and optimize the mosaicking flow on the basis of advanced parallel computing technology, the efficiency of parallel mosaicking will be improved.

In this study, we asynchronously processed the overlapping areas and non-overlapping areas in the mosaicking algorithm based on read filtering, and then we performed full-step mosaicking processing on the overlapping areas, while only performing image registration and radiometric normalization on the non-overlapping areas, which was expected to greatly reduce the cluster data I/O and computing load, thereby accelerating the mosaicking of massive remote sensing images. Figure 14 shows that the parallel mosaicking algorithm proposed in this paper is feasible. The results of Experiment 1, as shown in Figures 15 and 16 and Tables 2 and 3, show that the Spark-based parallel mosaicking algorithm is more efficient than MPI, and that the mosaicking algorithm proposed in this paper has less running time and higher throughput than state-of-the-art algorithms. The results of Experiment 2 (Figure 17 and Table 4) show that the mosaicking algorithm proposed in this

paper has good scalability. In summary, we can say that it is effective to accelerate parallel mosaicking by optimizing the flow of mosaicking.

However, further improvements are still required. First of all, the image data selected in this paper come from the same sensor and are highly consistent, which makes it possible to construct a global polynomial based on mutual information. If the images to be mosaicked come from different sensors or have large differences, other image registration methods will need to be considered. Secondly, looking at the mosaic results in Figure 14, it can be seen that the transition between images still exists, and better seamline detection and image blending methods are needed in the future. Finally, we can see from Figure 17 that the speedup of Spark-RF is still far behind the linear speedup, so higher-performance parallel computing technology or further optimization of the mosaicking flow is needed.

5. Conclusions

This paper analyzed the characteristics of remote sensing image mosaicking and identified the deficiencies of current parallel mosaicking research. To address these challenges, a rapid mosaicking algorithm utilizing read filtering called Spark-RF was proposed. This algorithm initially partitions images into blocks and stores them in the MapFile, and then asynchronously reads the overlapping and non-overlapping area blocks based on the MapFileInputFormat. Subsequently, it executes the mosaicking process on the overlapping area, whereas the non-overlapping area undergoes only image registration and radiometric normalization. The experimental results revealed that our algorithm outperformed current techniques while maintaining the quality of the resulting image; the average throughput increased by 1.38 MB/S, and the average speedup increased by 0.87. This study provides a theoretical foundation and novel ideas for processing remote sensing images on cluster platforms. Future work could include further optimization of the mosaicking steps and exploration of parallel mosaicking on diverse cluster platforms.

Author Contributions: Conceptualization, P.N. and Z.C.; methodology, P.N.; software, Z.C.; validation, P.N., Z.C. and Y.W.; formal analysis, P.N.; investigation, Z.C.; resources, Y.W.; data curation, Y.W.; writing—original draft preparation, P.N.; writing—review and editing, P.N., Z.C. and Y.W.; visualization, Z.C.; supervision, Y.W.; project administration, Y.W.; funding acquisition, P.N. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by China Scholarship Council: 202102505004; Scientific Research Project funded by Hunan Provincial Department of Education: 21C0291; Research Start-up Foundation of the University of South China: 200XQD036.

Data Availability Statement: The Landsat 8 dataset involved in this paper can be downloaded at https://earthexplorer.usgs.gov/, and the detailed description of the dataset can be found in Section 3.1.

Acknowledgments: The authors would like to sincerely thank the editors and the anonymous reviewers.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Burt, P.J.; Adelson, E.H. A Multiresolution Spline with Application to Image Mosaics. *ACM Trans. Graph. TOG* **1983**, *2*, 217–236. [CrossRef]
2. Foody, G.M. Remote Sensing of Tropical Forest Environments: Towards the Monitoring of Environmental Resources for Sustainable Development. *Int. J. Remote Sens.* **2003**, *24*, 4035–4046. [CrossRef]
3. Joyce, K.E.; Belliss, S.E.; Samsonov, S.V.; McNeill, S.J.; Glassey, P.J. A Review of the Status of Satellite Remote Sensing and Image Processing Techniques for Mapping Natural Hazards and Disasters. *Prog. Phys. Geogr.* **2009**, *33*, 183–207. [CrossRef]
4. Hame, T.; Salli, A.; Andersson, K.; Lohi, A. A New Methodology for the Estimation of Biomass of Coniferdominated Boreal Forest Using NOAA AVHRR Data. *Int. J. Remote Sens.* **1997**, *18*, 3211–3243. [CrossRef]
5. Hansen, M.C.; Loveland, T.R. A Review of Large Area Monitoring of Land Cover Change Using Landsat Data. *Remote Sens. Environ.* **2012**, *122*, 66–74. [CrossRef]
6. Li, X.; Hui, N.; Shen, H.; Fu, Y.; Zhang, L. A Robust Mosaicking Procedure for High Spatial Resolution Remote Sensing Images. *ISPRS J. Photogramm. Remote Sens.* **2015**, *109*, 108–125. [CrossRef]

7. Li, X.; Feng, R.; Guan, X.; Shen, H.; Zhang, L. Remote Sensing Image Mosaicking: Achievements and Challenges. *IEEE Geosci. Remote Sens. Mag.* **2019**, *7*, 8–22. [CrossRef]
8. Feng, R.; Du, Q.; Li, X.; Shen, H. Robust Registration for Remote Sensing Images by Combining and Localizing Feature-and Area-Based Methods. *ISPRS J. Photogramm. Remote Sens.* **2019**, *151*, 15–26. [CrossRef]
9. Suzuki, T.; Amano, Y.; Hashizume, T. Vision Based Localization of a Small UAV for Generating a Large Mosaic Image. In Proceedings of the SICE Annual Conference 2010, Taipei, Taiwan, 18–21 August 2010; IEEE: Piscataway, NJ, USA, 2010; pp. 2960–2964.
10. Kim, D.-H.; Yoon, Y.-I.; Choi, J.-S. An Efficient Method to Build Panoramic Image Mosaics. *Pattern Recognit. Lett.* **2003**, *24*, 2421–2429. [CrossRef]
11. Hua, Z.; Li, Y.; Li, J. Image Stitch Algorithm Based on SIFT and MVSC. In Proceedings of the 2010 Seventh International Conference on Fuzzy Systems and Knowledge Discovery, Yantai, China, 10–12 August 2010; IEEE: Piscataway, NJ, USA, 2010; Volume 6, pp. 2628–2632.
12. Zhong, C.; Xu, Q.; Li, B. Relative Radiometric Normalization for Multitemporal Remote Sensing Images by Hierarchical Regression. *IEEE Geosci. Remote Sens. Lett.* **2015**, *13*, 217–221. [CrossRef]
13. Li, L.; Yao, J.; Xie, R.; Li, J. Edge-Enhanced Optimal Seamline Detection for Orthoimage Mosaicking. *IEEE Geosci. Remote Sens. Lett.* **2018**, *15*, 764–768. [CrossRef]
14. Wang, W.; Ng, M.K. A Variational Method for Multiple-Image Blending. *IEEE Trans. Image Process.* **2011**, *21*, 1809–1822. [CrossRef] [PubMed]
15. Ma, Y.; Chen, L.; Liu, P.; Lu, K. Parallel Programing Templates for Remote Sensing Image Processing on GPU Architectures: Design and Implementation. *Computing* **2016**, *98*, 7–33. [CrossRef]
16. Ma, Y.; Wang, L.; Zomaya, A.Y.; Chen, D.; Ranjan, R. Task-Tree Based Large-Scale Mosaicking for Massive Remote Sensed Imageries with Dynamic Dag Scheduling. *IEEE Trans. Parallel Distrib. Syst.* **2013**, *25*, 2126–2137. [CrossRef]
17. Jing, W.; Huo, S.; Miao, Q.; Chen, X. A Model of Parallel Mosaicking for Massive Remote Sensing Images Based on Spark. *IEEE Access* **2017**, *5*, 18229–18237. [CrossRef]
18. Eken, S.; Mert, Ü.; Koşunalp, S.; Sayar, A. Resource-and Content-Aware, Scalable Stitching Framework for Remote Sensing Images. *Arab. J. Geosci.* **2019**, *12*, 1–13. [CrossRef]
19. Chen, L.; Ma, Y.; Liu, P.; Wei, J.; Jie, W.; He, J. A Review of Parallel Computing for Large-Scale Remote Sensing Image Mosaicking. *Clust. Comput.* **2015**, *18*, 517–529. [CrossRef]
20. Wang, Y.; Ma, Y.; Liu, P.; Liu, D.; Xie, J. An Optimized Image Mosaic Algorithm with Parallel Io and Dynamic Grouped Parallel Strategy Based on Minimal Spanning Tree. In Proceedings of the 2010 Ninth International Conference on Grid and Cloud Computing, Nanjing, China, 1–5 November 2010; IEEE: Piscataway, NJ, USA, 2010; pp. 501–506.
21. Merzky, A.; Stamou, K.; Jha, S.; Katz, D.S. A Fresh Perspective on Developing and Executing DAG-Based Distributed Applications: A Case-Study of SAGA-Based Montage. In Proceedings of the 2009 Fifth IEEE International Conference on e-Science, Oxford, UK, 9–11 December 2009; IEEE: Piscataway, NJ, USA, 2009; pp. 231–238.
22. Berriman, G.B.; Laity, A.C.; Good, J.C.; Katz, D.S.; Jacob, J.C.; Deelman, E.; Singh, G.; Su, M.-H.; Prince, T.A. Science Applications of the Montage Image Mosaic Engine. *Proc. Int. Astron. Union* **2006**, *2*, 621. [CrossRef]
23. Wang, L.; Ma, Y.; Zomaya, A.Y.; Ranjan, R.; Chen, D. A Parallel File System with Application-Aware Data Layout Policies for Massive Remote Sensing Image Processing in Digital Earth. *IEEE Trans. Parallel Distrib. Syst.* **2014**, *26*, 1497–1508. [CrossRef]
24. Zaharia, M.; Chowdhury, M.; Franklin, M.J.; Shenker, S.; Stoica, I. Spark: Cluster Computing with Working Sets. In Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10), Boston, MA, USA, 22 June 2010.
25. Wu, Y.; Ge, L.; Luo, Y.; Teng, D.; Feng, J. A Parallel Drone Image Mosaic Method Based on Apache Spark. In *Cloud Computing, Smart Grid and Innovative Frontiers in Telecommunications, Proceedings of the 9th EAI International Conference, CloudComp 2019, and 4th EAI International Conference, SmartGIFT 2019, Beijing, China, 4–5 December and 21–22 December 2019*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 297–311.
26. Ma, Y.; Song, J.; Zhang, Z. In-Memory Distributed Mosaicking for Large-Scale Remote Sensing Applications with Geo-Gridded Data Staging on Alluxio. *Remote Sens.* **2022**, *14*, 5987. [CrossRef]
27. Zitova, B.; Flusser, J. Image Registration Methods: A Survey. *Image Vis. Comput.* **2003**, *21*, 977–1000.
28. Maes, F.; Collignon, A.; Vandermeulen, D.; Marchal, G.; Suetens, P. Multimodality Image Registration by Maximization of Mutual Information. *IEEE Trans. Med. Imaging* **1997**, *16*, 187–198. [PubMed]
29. Pass, G.; Zabih, R. Comparing Images Using Joint Histograms. *Multimed. Syst.* **1999**, *7*, 234–240. [CrossRef]
30. Mills, A.; Dudek, G. Image Stitching with Dynamic Elements. *Image Vis. Comput.* **2009**, *27*, 1593–1602. [CrossRef]
31. Han, X.; Cao, H.; Yuan, Z.; Zhao, H.; Yan, L. An Approach of Color Image Mosaicking Based on Color Vision Characteristics. In Proceedings of the 2009 Third International Conference on Genetic and Evolutionary Computing, Guilin, China, 14–17 October 2009; IEEE: Piscataway, NJ, USA, 2009; pp. 343–346.
32. Fernandez, E.; Garfinkel, R.; Arbiol, R. Mosaicking of Aerial Photographic Maps via Seams Defined by Bottleneck Shortest Paths. *Oper. Res.* **1998**, *46*, 293–304. [CrossRef]
33. Kass, M.; Witkin, A.; Terzopoulos, D. Snakes: Active Contour Models. *Int. J. Comput. Vis.* **1988**, *1*, 321–331. [CrossRef]
34. Dijkstra, E.W. A Note on Two Problems in Connexion with Graphs. In *Edsger Wybe Dijkstra: His Life, Work, and Legacy*; ACM: New York, NY, USA, 2022; pp. 287–290.

35. Agrawal, H.; Horgan, J.R. Dynamic Program Slicing. *ACM SIGPlan Not.* **1990**, *25*, 246–256. [CrossRef]
36. Sheoran, S.; Sethia, D.; Saran, H. Optimized Mapfile Based Storage of Small Files in Hadoop. In Proceedings of the 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), Madrid, Spain, 14–17 May 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 906–912.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.