International Journal of Geo-Information

MDPI

Article

An MPI + OpenMP Hybrid Parallel Paradigm for Pyramid Building Algorithms of Large-scale Remote Sensing Images

Shiyong Liu [1,†], Luo Chen [2,*] and Jun Li [3]

1 National University of Defence Technology; shiyongliu@nudt.edu.cn
2 National University of Defence Technology; luochen@nudt.edu.cn
3 National University of Defence Technology; junli@nudt.edu.cn
* Correspondence: luochen@nudt.edu.cn; Tel.: +86-139-7580-1542
† Current address: No.109, Deya road, Changsha, Hunan, 410073, China

Academic Editor: name
Received: date; Accepted: date; Published: date

Abstract: To speed up the visualization of remote sensing image and improve the fluency of multi-level browsing in both Client/Server and Browser/Server, the application of pyramid building method is an indispensable choice. Pyramid building algorithms are data and computation intensive. With the development of modern remote sensing technology, the scale of remote sensing data obtained is becoming larger and larger. The processing performance can be poor if traditional sequential processing techniques are adopted to these large-scale image data. Based on the advantage of high-performance parallel computing, this paper presents a MPI and OpenMP based hybrid parallel computing paradigm, named "ParaOvr", for pyramid building. To realize fast data reading and writing without conflicts, this paper gives a solution by pre-planning the data organization of an image pyramid. The proposed data organization scheme overcomes the limits of pyramid file, which must be written by a fixed size for one time. Furthermore, the same level of pyramid data is compact organized. In order to improve the efficiency of image processing, this paper presents a data-partition method based on line division. In this method, the number of data partitions is equal to the total number of processes during image processing. Each process includes reading, writing and re-sampling parallel to its data. The experiments show that compared to ArcGIS and GDAL, ParaOvr has a super linear acceleration in data processing. The advantages become more obvious when the size of data is large.

Keywords: parallel processing; remote sensing; image pyramid; MPI/OpenMP

1. Introduction

Image pyramid is a multi-resolution raster data structure. In simple terms, the structure of an image pyramid is a serie of different resolution raster images, established from the original raster image, each image resolution corresponds to each pyramid level [1]. At the same time, pyramid is also a raster image loss compression. When the user needs an operation such as "zoom in", "zoom out" or "pan" to obtain raster images with different resolutions, system can select a similar resolution of data according to the user view for visualization. So the system only needs a few calculation and query to obtain the results without any sequential sampling calculation. Therefore, image pyramid can reduce the time of data display and improve the raster image visualization performance [2,3].

Since pyramid model is an effective method for fast image display and efficient processing [4,5], pre-using pyramid model to generate multi-level resolution images has become a premise and foundation for the release of remote sensing images on the Internet and many other fields such as

pattern recognition, Graph Signals Transformation, and Image Change Detection. Ren proposes three standard approaches to build irregular pyramid partitions for image retrieval the bag-of words model [6]. Yan *et al*. propose a accurately in two steps method to extract lines based on a binary image pyramid and Hough transforms [7]. Teng *et al*. propose an image enhancement method based on Laplacian pyramid [8]. Zhao *et al*. propose a novel fast haze removal technique for single image using Image Pyramid [9]. Li constructs the multi-resolution scheme using a Gaussian image pyramid [10]. Belle presents a method using image pyramid to reduce the cost of images capturing [11]. Momeni, uses image pyramids to build up a face recognition system [12]. Wei *et al*. propose a fast view scheme for mass remote sensing images based on image pyramid [3]. Huang uses an image pyramid for image segmentation [13]. Since image pyramid is widely used in various fields, yet remote sensing images contain more and more large volumes of data and complex information [14]. It is more difficult for traditional sequential processing algorithm and commercial software to deal with such large-scale data efficiently. Therefore, image pyramid generation has become a key problem for the high efficiency management and visualization of remote sensing image [15].

Currently, research on improving the efficiency of pyramid building is mainly focused on two aspects, the first one is that improvement on existing pyramid model, another is parallelization of pyramid [16]. Cheng *et al*. uses global subdivision grid to improve the traditional image pyramids [17]. In order to ensure the generality of the pyramid file, modification of existing models is difficult to significantly improve the efficiency of production in a pyramid. Personal computers can hardly fast handle large-scale image processing task efficiently due to the limitation of hardware, with the development of computer technology, computing resources become more and more. Therefore, using the parallel processing mechanism of multiprocessing and multi-node to accelerate the processing speed of geospatial data has become an inevitable trend [18–21]. Zheng *et al*. propose a distributed method to accelerate the pyramid building speed [22], in the method he split the traditional task into multiple grid cells according to the principle of equal tile, then let each grid cell as input data, finally begin the parallel pyramid construction process, but this method is complex for multi node task division and boundary grid. Kang *et al*. presents a method to parallelly build image pyramid based on CUDA [16]. However, this system architecture has a high hardware cost, it needs to read data constantly from the host memory into the GPU memory, and it cannot take the advantage of clusters. Yi *et al*. proposes an opinion that uses MapReduce to build remote sensing image pyramid, this method can use parallel disk system and cluster to handle large-scale remote sensing images [23], but this method requires the data to be distributed storage and the results collection, and this is usually time-consuming. He *et al*. use Message Passing Interface (MPI) to build pyramid of large image datasets, and the result is good [24], but he did not make a further research on the structure of pyramid. In order to get the strip-offset of each process, some data need to be written into each strip in advance using IO function of GDAL library, this is a time cost procedure, furthermore the BIL storage mode for improving the efficiency of parallel IO is also limited, and with the increases data amount, the expenses of creating file view will becomes larger and larger.

In order to speed up the algorithm, this paper proposes an approach named "ParaOvr". It uses the shared-memory to build a pyramid in parallel. ParaOvr takes the hybrid parallel strategy of multiprocessing and multithreading based on MPI and OpenMP (Open Multi-Processing). If multiple processes cannot meet the application demands, the algorithm will be supplemented by multiple threads, the results show that this method can greatly improve processing efficiency and stability. Through study of the data structure of image pyramid, this paper proposes a highly efficient and stable parallel access technology in pyramid file to pre-planning pyramid file data structure before creating an empty pyramid file, so each process can be accurately read and written on the pyramid files. Using ParaOvr, each process does not have to be under a fixed strip write tape size, it also solves the common black edge and fragmented IO problem. A substantial increase in the parallel IO speed and an improved pyramid construction efficiency have been achieved by ParaOvr.

2. Background

Image pyramid is stored in a single file. There are two types of file formats respectively, overview (".ovr") and the reduce resolution dataset (".rrd "). The rrd format is adopted by ArcGIS before the version of 10.0, and is taken over by the format ovr after version 10.0. Ovr format is now the current mainstream format to store the pyramid layer of raster dataset. Compared to the rrd format, ovr allows data compression to control the quality of pyramid [25]. Based on the above characteristics, this paper adopts the ovr format pyramid, ovr file uses the same data structure as the TIFF file, and its file structure mainly contains the following three parts: Image File Header(IFH), Image File Directory(IFD), Image Data(DATA) [26]. See Figure 1.

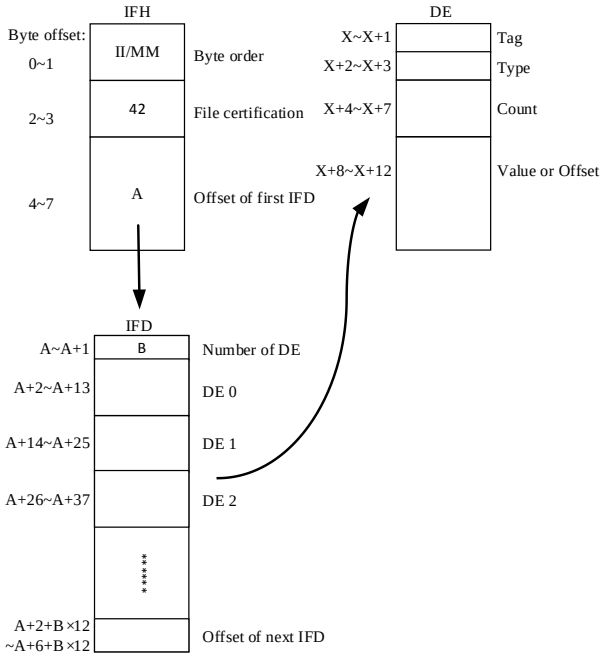

Figure 1. Structure of ovr format pyramids files

As shown in Table 1, IFH total size is 8-byte, recording the byte order of the file, the ovr(TIFF) certification number, and the first IFD offset. Bytes 0-1: the byte order used within the file, in the "II" format, byte order is always from the least significant byte to the most significant byte. In the "MM" format, byte order is always from most significant to least significant. All for both 16-bit and 32-bit integers. Bytes 2-3: an arbitrary chosen number 42 that further identifies the file as an ovr (TIFF) file. Bytes 4-7: offset of the first IFD. Particularly, Image File Directory (IFD) may follow the image data it describes [26].

Table 1. IFH data structure

Offset	Description	Value
Bytes 0-1	Byte order flag	II or MM
Bytes 2-3	ovr(TIFF) file identification	42
Bytes 4-7	The offset (in bytes) of the first IFD	Any location but must begin on a word boundary

IFD contains four parts: 2-byte for directory entry number of current pyramid level (DEC), a sequence of 12-byte each directory entry, 4-byte offset of offset to next IFD, there must be at least 1 IFD in a TIFF file and each IFD must have at least one entry. See Table 2.

Table 2. IFD data structure

Name	Size in byte	Description
DEC	2	The number of values
DE	112	The first IFD Entry (DE)
DE	212	The second IFD Entry (DE)
.....	.....	.....
DE	n12	The nth IFD Entry (DE)
NIFD	4	Next offset of IFD relative to pyramid file beginning

A TIFF field is a logical entity consisting of TIFF tag and its value. This logical concept is implemented as an image file Directory Entry (DE), the number of directory entries can be different, users can extend the type and number of DE according to their needs. Each DE describes a field of current pyramid level. The DE consists of four parts: TIFF Tag, Field Type, Count and Value. The TIFF Tag is an integer number to mark the name of the field. For example, Tag 262 represents PhotometricInterpretation label. Tag 257 represents ImageLength and 256 represents ImageWidth. All entries in an IFD must be sorted in ascending order of Tag. Type describes the size of the field, such as BYTE, ASCII, SHORT and LONG. Count is the number of values not the total number of bytes. To improve the computational and space efficiency, the value offset contains the value instead of pointer of memory address only if the value is fitted into 4 bytes. If the value is shorter than 4 bytes, it is left-justified within the 4-byte value Offset. The Type and Count of the field determine whether the Value can be represented within 4 bytes. Table 3 gives a short brief about basic image file directory entries [26].

Table 3. Example of IFD Entry

Tag	Decimal	Type	Value
ImageWidth	256	SHORT/LONG	The number of rows in the image
ImageLength	257	SHORT/LONG	The number of columns in the image
Compression	259	SHORT	1 = No compression 2 = Huffman encoding 32773 = PackBits compression
StripOffsets	273	SHORT/LONG	The byte offset of each strip
RowsPerStrip	278	SHORT/LONG	The number of rows in each strip
XResolution	282	RATIONAL	The number of pixels per ResolutionUnit in the ImageWidth
YResolution	283	RATIONAL	The number of pixels per ResolutionUnit in the ImageLength
ResolutionUnit	296	SHORT	1 = No absolute unit of measurement 2 = Inch 3 = Centimeter

The data in pyramid are organized by strips or tiles, position of each strip or tile is defined by StripOffset field. Furthermore, StripOffset field is the only one-way to access to the data offset. So based on StripOffset field, each data strip of tile in pyramid can be written into anywhere except where IFH and IFD are located. Yet this raise a new problem that two logically adjacent blocks may not be adjacent to physical storage. As Figure 2 shows, two logically adjacent strips of 3 bytes are stored in the physical pyramid file.

3. Hybrid Parallel Paradign for Pyramid Building of Remote Sensing Data

3.1. Pyramid's Image Data Organization of ParaOvr

Based on aforementioned reasons, it is necessary to pre-organize the storage location of the pyramid before writing, so that the same level of pyramid data can be physically contiguous storaged, and conflict with the location of IFH and IFD can be avoided.

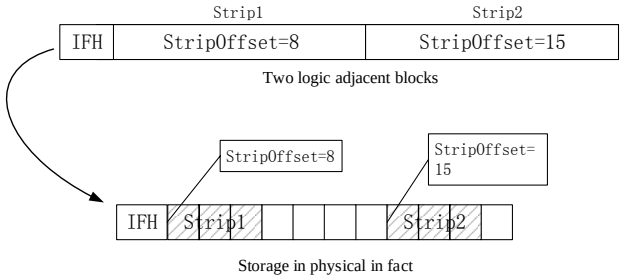

Figure 2. Pyramid data storage in physical

In order to let the pyramid's image data stored contiguously, this paper uses the storage of IFH-DATA-IFD. In Figure 3, take the first level of the pyramid for example, assuming the width of current level of the pyramid is *XSize*, the height is *YSize*, the bands count is *nBands*, number of directory entry is *DEC*. The calculation formula of the image data size of the current level is as follows:

$$imgSize = XSize \times YSize \times nBands \tag{1}$$

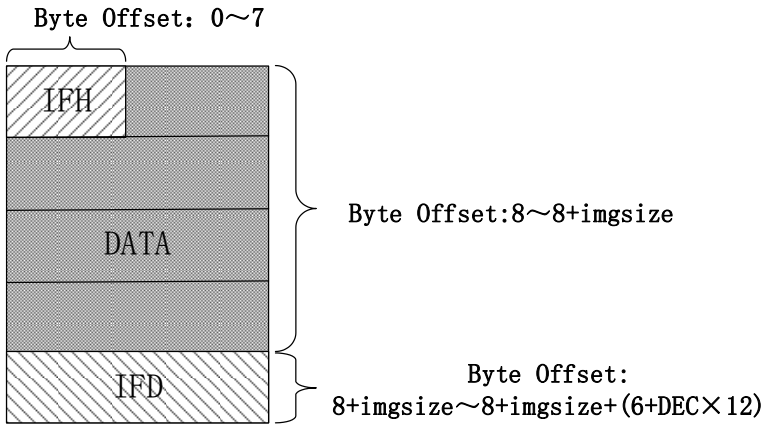

Figure 3. Pyramid files structure

The strategy of writing resampling data into a pyramid file is as follows: first, write flag information of IFH data at the 0∼3 byte offset of the beginning of the pyramid file (as described in Table 1), then write the first IFD offset value at the offset 4∼7 byte, the value of the first IFD offset can be calculated with the following formula Equation (2).

$$IFDoffset = 8 + imgSize \tag{2}$$

The image data byte offset range is 8∼IFDoffset. Finally, when the image data writing process is over, we begin to write the IFD. IFD size can be described by Equation (3).

$$IFDsize = 2 + DEC \times 12 + 4 \tag{3}$$

Meanwhile, the value of StripOffset field of strip m is given in Equation (4). The *StripSize* is calculated as the size of each strip in byte.

$$StripOffset = 8 + m \times StripSize \tag{4}$$

The data structure of other level of the pyramid is similar to the first level, assuming that the total level of the pyramid is n. The initial data offset of level *lev* can be calculated as Equation (5).

$$DataOff(lev) = \begin{cases} 8 & lev = 0 \\ DataOff(lev-1) + ImgIFD & lev > 0 \end{cases} \tag{5}$$

$$ImgIFD = Imgsize + IFDsize \tag{6}$$

Assuming that *StripOffset* is the offset of strip m in level *lev*, then *StripOffset* can be calculated as Equation (7).

$$StripOffset(lev, m) = DataOff(lev) + m \times StripSize \tag{7}$$

3.2. Data Partitioning Strategy of ParaOvr

Data partitioning for each process is an important step during parallel image pyramid building. The performance of the final algorithm is directly determined by the task assignment. Because line division has the highest efficiency on allocation and execution [27,28], ParaOvr adopts the line division method to perform data partitioning. The detailed description is shown in Figure 4, assuming the original image width is *imgXsize*, the height is *imgYsize*, the total number of process involved in the operation is n, each process is described as $Rank_i$ (i=0, …, n-1). So every band is divided into n strips, each $Rank_i$ map to one $Strip_i$ (i=0, …, n-1) respectively. Assuming that the size of each process $Rank_i$ is *srcBuffsize*, we can get the *srcBuffsize* by Equation (8):

$$\begin{aligned} srcBuffsize &= srcBuffXsize \times srcBuffYsize \\ srcBuffXsize &= imgXsize \\ srcBuffYsize &= \begin{cases} imgYsize/n & i < n-1 \\ ImgYsize - [(n-1) \times \frac{imgYsize}{n}] & i = n-1 \end{cases} \end{aligned} \tag{8}$$

The reading start offset of current band of process $Rank_i$ can be got by Equation (9):

$$offset(i) = i \times (imgYsize/n) \tag{9}$$

Each process for reading image data to memory parallelly from the original image is related to *offset(i)*, then each process resamples its data according to pyramid level, finally, the resampling data is written to the corresponding location of pyramid file.

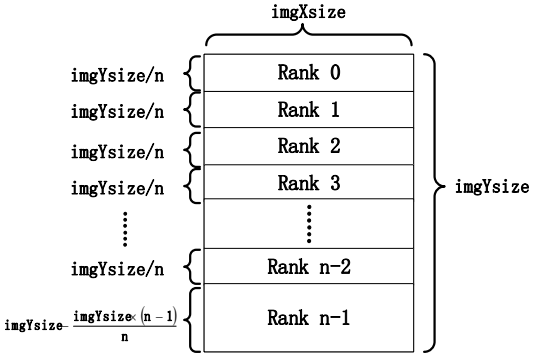

Figure 4. Image data partitioning in each process

It may have a slight problem using the above task partitioning method. If the image size is too large and the number of processes is not enough, then the data size assigned to each process maybe too large, and if the data size is larger than the upper bound of integer, the function RasterIO of GDAL

is not able to handle such a big data, may cause type overflow. We propose a solution combined with multithreading and multiprocessing in this paper to address the issues discussed above. As shown in Figure 5, when the data size of a process reaches an integer upper bound, the Open Multi-Processing (OpenMP) is used to multithread this process, then the multiple threads are used to segment the data of the process $Rank_i$ again.

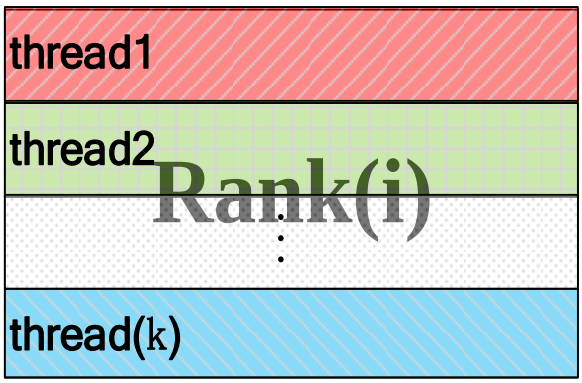

Figure 5. Using multithreading to divide data of process

Assuming that the data size of current process $Rank_i$ is *rankSize*, the upper limit of integer is INT_MAX, the number of threads used in process $Rank_i$ is k (k is an integer), then we can get k by Equation (10):

$$\frac{rankSize}{2^k} \leqslant INT\_MAX \leqslant \frac{rankSize}{2^{k-1}} \tag{10}$$

The detail procedure of multithreading division method is described as the pseudo codes in Table 4, the effect of function GDALRasterIO is to read the image data from a source remote sensing image. Each thread according to the logical offset of their parent process and the relative offset to their parent, then we can calculate the logical offset position from the source raster image of each thread. Because all the threads are sharing their common parent process memory address space, each thread can use the GDALRasterIO function of GDAL library to read image data in parallel and independently from the memory space of their parent process, according to the logical offset position from the source raster image we have calculated.

Table 4. Pseudo-codes of data partition based on multiprocessing and multithreading

```
1  if(rankSize > INT_MAX) {
2    int k = 1;
3    while(rankSize > INT_MAX) { //k presents as the total number of threads
4      k = k*2;
5      rankSize = rankSize/2;
6    }
7    long nSubYsize = srcBuffsize/k;
8    #pragma omp parallel for private(j) //begin multi-thread processing.
9    for(int j=0; j<k; j++) { //each thread read data to their parent process
10       GDALRasterIO(GF_Read, 0, offset(i) + j*nSubYsize, imgXsize,...);
11   }
12 }
```

3.3. Data Parallel Resampling

In this paper, we use the nearest neighbor resampling method for each process or thread to resample its reading data [29,30]. In simple terms, the length and width takes a pixel for each 2^{lev}

pixel, *lev* is presented as current level of pyramid. Because of the boundary problem, the adjacent process which has the same amount of reading data may not always have the same final resampling result. Figure 6 shows the schematic diagram of parallel resampling of each process in this paper, in this figure we use a 7×11 image data for example to perform the sampling operation of first pyramid layer by process number n=3, the black block is the sampling results. From Figure 6, we can see that the resampling sized of Rank0, Rank1, and Rank2 is 2, 1, 3, respectively. *RowOffset* for the starting offset of black block within each process, meet Rank0 is 0, Rank1 is 1, Rank2 is 0, therefore, the data resampling of each process should from the line offset *RowOffset* not just simply operate from the data start position.

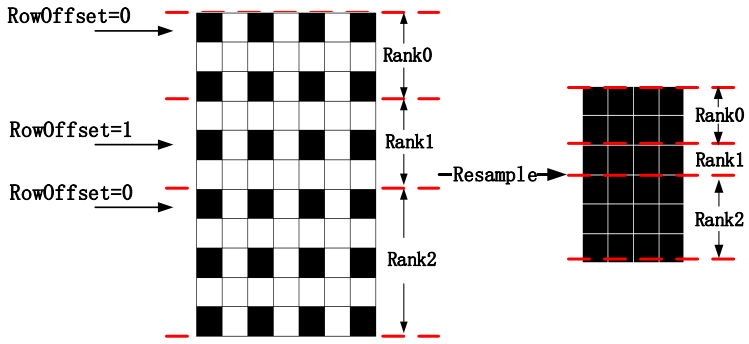

Figure 6. Data parallel resampling process

The size of the resampling data is not always all the same because of the inconsistency of *RowOffset*, *RowOffset* is the sampling starting position of each process. *RowOffset(i,m)* is the *RowOffset* of each process in $Rank_i$ and level of *lev*, and can be calculated by Equation (11):

$$RowOffset(i,lev) = \begin{cases} 0 & offset(i)\%2^{lev} = 0 \\ 2^{lev} - offset(i)\%2^{lev} & offset(i)\%2^{lev} \neq 0 \end{cases} \tag{11}$$

The *offset(i)* is the reading start offset of the current band of process $Rank_i$ defined by Equation (9). The size of resampling data of each process $Rank_i$ in level of *lev* can be calculated using Equation (12):

$$\begin{aligned} resBuffsize(i,lev) &= resBuffX(i,lev) \times resBuffY(i,lev) \\ resBuffX(i,lev) &= \frac{imgXsize}{2^{lev}} \\ resBuffY(i,lev) &= \frac{imgYsize - RowOffset(i,lev)}{2^{lev}} \end{aligned} \tag{12}$$

Based on the parallel resampling method above. Each process only needs to read the raw data from a hard drive to memory once. When performing resampling process, we need to set different sampling intervals and then reduce resampling from the original data.

3.4. Data Parallel Writing Method

As shown in Figure 7, pyramid data writing is divided into two parts: sequential and parallel. For sequential process, Rank0 creates an empty pyramid file using the proposed IFH-DATA-IFH structure, set aside the DATA space, and writing IFH and IFD into the appropriate location of the empty pyramid using Libtiff library.

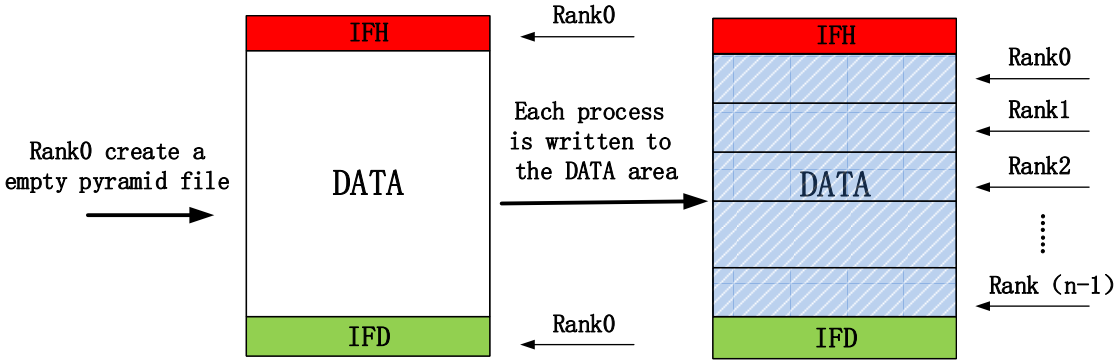

Figure 7. Resampling data of pyramid writing method

Once IFH and IFD have been written into the empty pyramid file, then the parallel process begins. We adopt the MPI parallel IO interface to write the resampling data of each process into the DATA region in parallel. He *et al*. propose a MPI parallel writing method based on the File View and Band Interleaved by Pixel Format (BIP) storage strategy [24]. In Figure 8, we take an image with 3 bands for example, to have a brief explanation to BIP data organization, as the figure shows, images are stored sequentially in the order of all bands of the first image pixel, then storing all bands of the second image pixel, cross storage and so on until all pixels are done [25]. Because all the data between the same band are not contiguous, single-band and multi-band should to be considered separately. In particular, you need to create a file view to deal with multi-band image which is a very complex task.

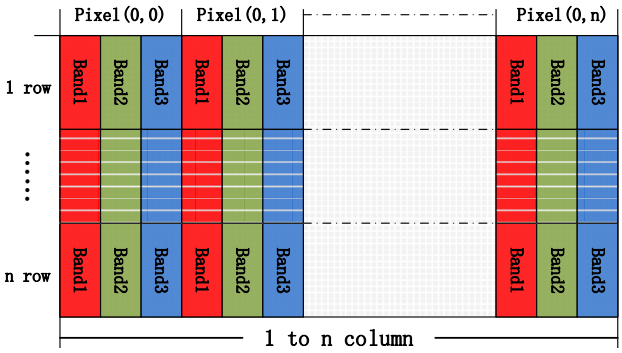

Figure 8. BIP data organization

For the lack of BIP data organization in parallel IO, this paper proposes a new parallel data storage structure using Band Sequential Format (BSQ) [25] which overcomes the shortcomings of the BIP method of IO discontiguous and the complexity of file view creating. The main idea of the algorithm is that each row of data are stored closely followed by the next row data of the same band, after one band is done then save the closely followed next band (Figure 9). Since all the data in the same band are stored contiguously, we do not need to create File View nor process single-band and multi-band separately, we just need to calculate the absolute offset in the pyramid of each process before writing.

With the band division method, we can get the absolute offset *AbsOff* of pyramid in level *lev* in band q in $Rank_i$ using Equation (13):

$$\begin{aligned} AbsOff(lev,k,i) &= DataOff(lev) + OvrSize(lev,k,i) \\ OvrSize(lev,k,i) &= k \times XSize \times YSize + \sum_{j=0}^{i-1} resBuffsize(j,lev) \end{aligned} \tag{13}$$

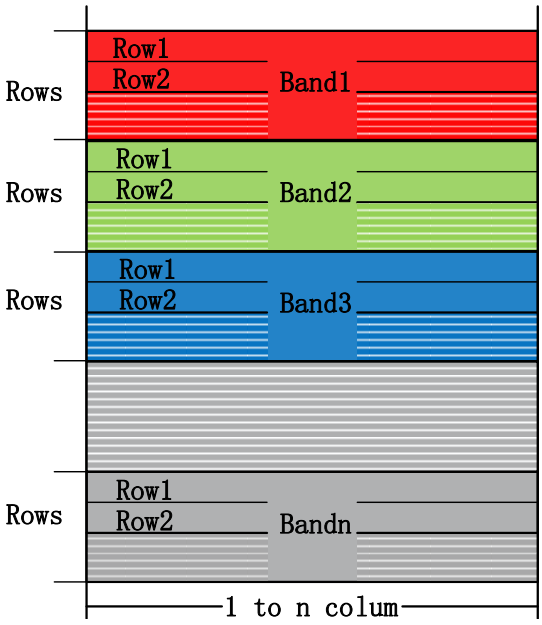

Figure 9. BSQ data organization

So we only need to put the absolute offset position *AbsOff*, and the size of resampling data buffers *resBuffsize* as arguments to the MPI_File_write_at() function to write the resampling data into pyramid files.

3.5. Algorithm execution process of ParaOvr

The flow chart of ParaOvr is shown in Figure 10, the detailed description is as follows:

1. Initialize pyramid level, total process number, and *lev*—the initial iterator of pyramid level is set to be zero. A number is assigned to each process, then, we can describe the subsequent operation of the process using the process number.
2. Each process reads the metadata information of the original image, such as width, height, bands count, and data type. Then, we define Rank0 as the master process, next step Rank0 create the empty pyramid file according to the above metadata and the IFH-DATA-IFD rule we have written above.
3. Dividing source image data averagely to each process by the line method, then for each process the logical offset position of their allocated data block is calculated according to the process rank number.
4. For each process, the data are read from the corresponding region of the original image to the memory according to the data partition rule. If the data size of each process is large than INT_MAX, then multiple threads are used to do a further division.
5. For each process, a 2^{lev} granularity nearest neighbor interpolation algorithm is used to resample the data read from step 5. *lev* is the current pyramid level.
6. Each process calculates the absolute write offset position in empty pyramid file according to process-number and band-number, then MPI_File_write_at() function is used to parallelly write the resampling data from memory into the empty pyramid file based on the obtained offset position.
7. Add the current Pyramid level to 1, if the pyramid level is fewer than or equal to the pyramid level we have set, then it to step 5. if the pyramid level is larger than the pyramid level we have set, the pyramid construction is completed.

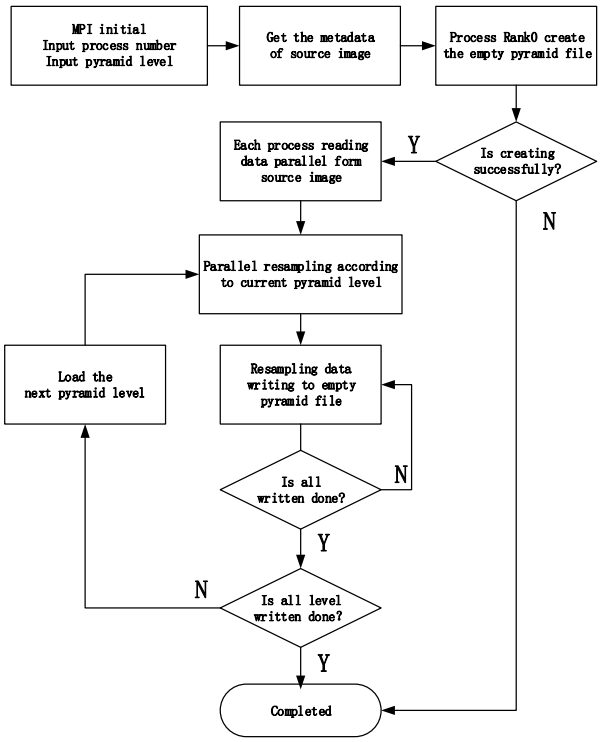

Figure 10. A flow chart of ParaOvr algorithm

4. Experimental Results and Analysis

Two Supermicro high performance computers are used in the experiments. One is with Linux CentOS6.3.X86_64 operation system (OS), another is with Windows 7 OS, and configurations of the computer are given in Table 5. The GCC 4.4.6 and MPICH 3.0.4 are used as the compiler and MPI, respectively.

Table 5. Configurations of the Supermicro computer

Type	Description
CPU	Four Inter(R) Xeon CPU E5-4620, 2.60GHz, 8 cores per CPU, Totally 64 virtual CPU cores
RAM	Totally 768GB, Samsung 32GB per each
File system	A disk array with 48TB
OS	Windows 7 Professional, 64-bit / Linux CentOS6.3.X86_64

Experimental test data with different size and band count are used in ParaOvr to build a 9-level pyramid, information on the images is given in Table 6.

Table 6. Configurations of the Supermicro computer

Name	Size[a]	Image Size[b]	Type	Bands
0.TIF	12.9	$72,001 \times 48,001$	Pan	1
1.TIF	14.2	$87,040 \times 58,368$	Multi	3
2.TIF	53.2	$220,672 \times 86,272$	Multi	3
3.TIF	115.9	$432,001 \times 144,001$	Pan	1
4.TIF	137.81	$136,448 \times 90,368$	Multi	3
5.TIF	278.2	428142×232572	Multi	3

a. The unit is *"GByte"* b. The unit is *"Pixel"*

4.1. Time consuming with respect to the number of processes

Six images shown in Table 6 are used in this experiment, and total time cost of the algorithm for each image is recorded. We uses the average value of ten experiments by removing the maximum and minimum values.

From Figure 11 we can see that the change of computational time of the algorithm varies with the number of processes, we can see from Figure 11 that: (1) In a certain range, the efficiency of the algorithm increases with the number of processes obviously, but if the number of processes increases further, the efficiency of algorithm gradually tends to be gentle, however if number of process is further increased, the efficiency of algorithm may be declined by a certain degree. (2) More data can produce the better efficiency performance. (3) For multi-band images, ParaOvr has the same high efficiency as a single-band image. The main reasons are that with the increase number of processes, task will be divided into more processes, the scale of the task for each process will be smaller, so the overall performance of the algorithm is improved. But when the number is increased to a certain degree, performance tend to be stable because of the limit of IO maximum speed of a hard disk. But the number of processes is further increased, IO competing may occur between each process, leading to a decline in performance. So a proper number of processes to different size of image are a better way to access the best performance.

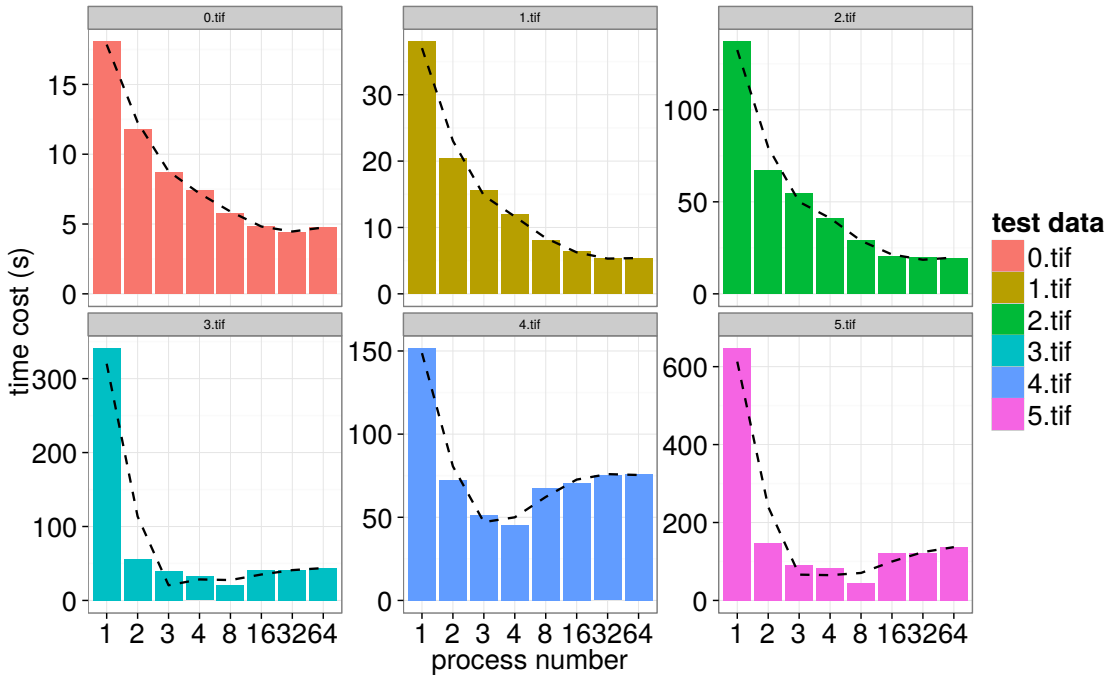

Figure 11. The parallel depth with different image size

There is a special phenomenon in Figure 11, in test data 3.tif and 4.tif we can see when the process is 1, the time cost 3.tif is larger than 4.tif though the file size of 4.tif larger than 3.tif. It is mainly because 3.tif is a single-image with image size of $432,001 \times 144,001$ which is far larger than $136,448 \times 90,368$ of 4.tif though the file size is not larger than 4.tif. When only one process is used in ParaOvr, in order to avoid type overflow, we need to read data from multitimes. This means more IO operation to a larger image size, so the time cost of 3.tif is larger than 4.tif. But this problem will gradually decrease with the number of process.

Another interesting observation is that, by comparing process 1 and process 2 in image 3.tif or image 5.tif, we can see that process 2 has a super linear acceleration as compared to process 1, this maybe because ParaOvr uses the line-partition method, 3.tif and 4.tif have one thing in common that

their data size of each line is very large, when using only one process, we have to read less lines for each time in order to avoid type overflow, so it has a great pressure in IO to finish such a large data. But when we use two processes, each process can also be divided into multiple threads, so each process can better elaborate the performance of parallel IO.

4.2. Comparasion with Open-Source GDAL Pyramid Algorithm

The GDAL (Geospatial Data Abstraction Library) is a powerful geography data conversion library, there is a pyramid building tool named "gdaladdo", we manually separate the six images above into two groups, the performance is tested in single-band and multi-band images between GDAL and ParaOvr. The average of ten experimental results is obtained by removing the maximum and minimum values. The details are shown in Figure 12.

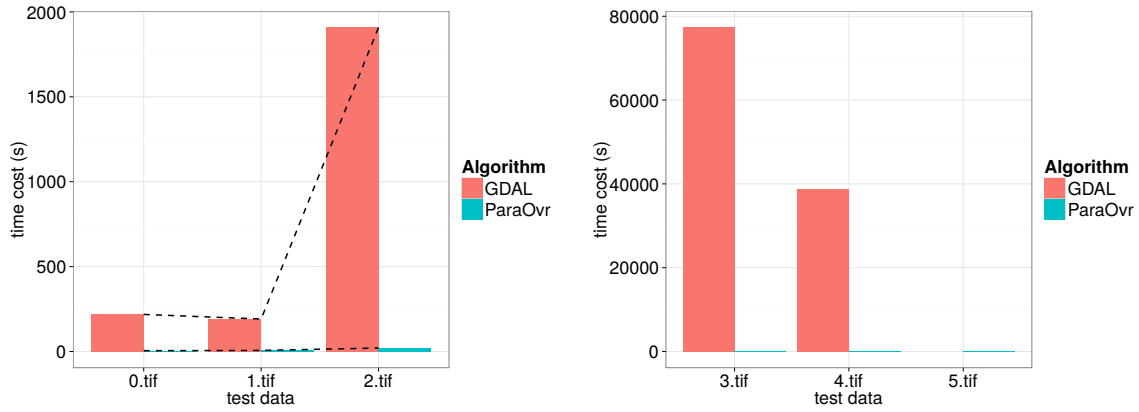

Figure 12. Performance between ParaOvr and gdaladdo

We can find from Figure 12 that a large size of test data produces poor performance. When the scale of dataset reaches 100GB, the efficiency of gdaladdo is unacceptable, and if the scale up to image 5.tif, the gdaladdo can no longer work after running 135 hours. You can see a huge performance boost from ParaOvr as compared to gdaladdo. The gdaladdo is a sequential algorithm, it cannot cross node nor using multi-core computing resources, and the writing method of fixed tile size mode will increase the IO times to some extent. Yet ParaOvr adopts the line mode writing method. Line mode can make better use of the computer's memory resources and reduce the IO times comparing with the fixed tile size mode. Each process is processing its corresponding data, which realizes the double parallel of the resampling and IO, so the promotion of performance of the ParaOvr is very obvious.

4.3. Comparasion with Commercial Software ArcGIS

Because ParaOvr running in the Linux environment and ArcGIS can only be installed in Windows operation system, we choose two Supermicro computers which have the same hardware environment, the version of ArcGIS is 10.2, and there is a pyramid building tool in its ArcToolbox module, it allows to set a parallel parameter factor to achieve multithreading processing. So we test the ParaOvr and ArcGIS in the same parallel factor 16, the only difference is that ParaOvr using 16 processes to parallel process yet ArcGIS using 16 threads. The result value is adopted by the average of ten times testing value which the maximum and minimum values are removed. The details are shown in Figure 13.

From the experimental results, we can see that the parallel effect is not obvious though the parallel parameter is set. Compared to GDAL, ArcGIS performs better than GDAL in single-band image, but it is also short of efficiency when dealing with larger-scale image. When the test case reaches in 6.TIF, ArcGIS crashes after running 75h, so the ArcGIS cannot handle the pyramid building

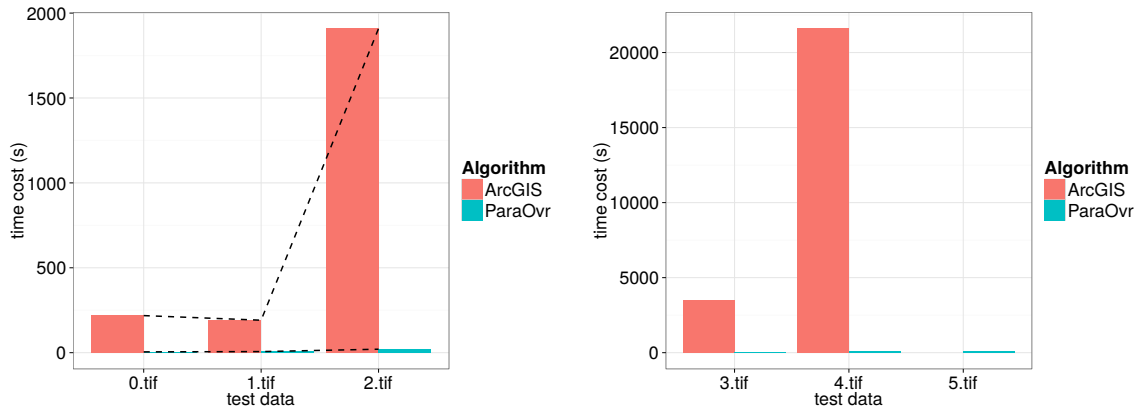

Figure 13. Performance between ParaOvr and ArcGIS

task of larger than 200GB, and ParaOvr has a better performance no matter in efficiency or stability with the increasing data size.

4.4. Comparasion with MPI based Pyramid Building Algorithm

He *et al.* proposed a pyramid parallel building algorithm using MPI[24], comparing with ParaOvr in process number 16, the experimental result is shown in Figure 14.

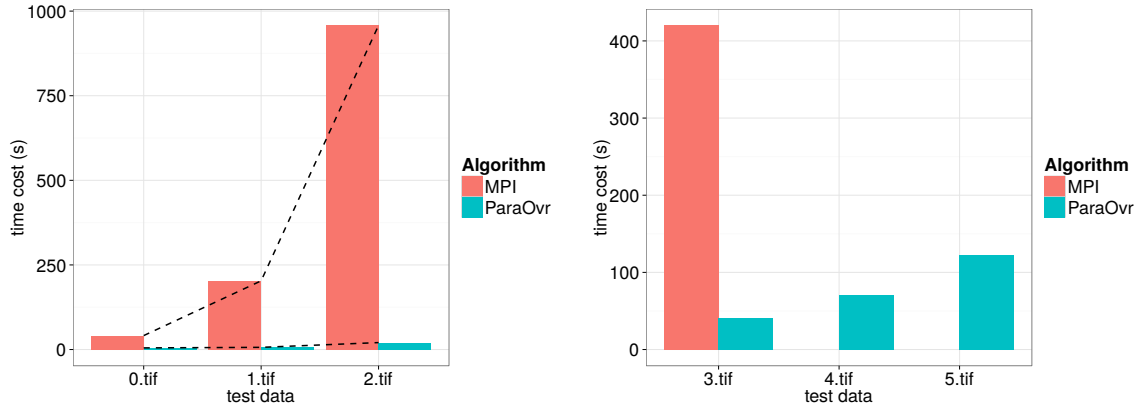

Figure 14. Performance between ParaOvr and MPI based algorithm

From Figure 14, we can see that the MPI based method can keep a high performance in processing single-band images, however, when facing multi-band images the performance will be sharply declined, this is more serious when the number of bands is increased. The main reason is that the MPI based method using file view writing based on BIL structure. The efficiency of this writing method is very low when dealing with multi-band images, from images 5.TIF and 6.TIF we can see that the MPI based method appears abnormal and we cannot get the experimental result. We can see that the efficiency of ParaOvr is roughly proportional to the MPI based method with the bands of image. ParaOvr can have the same performance as single-band when dealing with multi-band images. Furthermore, ParaOvr has a better performance when facing big data size.

5. Conclusions and Future Work

To address the problems and drawbacks of the current image pyramid building algorithms, this paper proposes a new parallel pyramid building method named "ParaOvr" based on MPI and OpenMP. ParaOvr improves the efficiency of parallel pyramid building process by pre-planning the data organization mode of image pyramid file and two level parallelism of resampling and IO.

During the parallel computing, the line data partitioning method is studied and a new parallel data reading method is proposed based on multiprocessing and multithreading to solve the type overflow problem, then we give the implementation method of parallel resampling and its calculation example. Finally, we propose a parallel writing method based on BSQ data storage model to solve the problem of low efficiency of multi-band images and the phenomena of data loss. The experimental result shows that the performance of ParaOvr has a great improvement compared to current methods and tools, especially the advantage is more obvious with the increase of the data size and the number of bands. Furthermore, the performance of ParaOvr can have a further boost through the improvement of the extension of file system bandwidth.

This work suggests several areas for future research: One interesting issue for future work is using distributed parallel computing platform like Spark to implement the algorithm. In this paper, we use the centralized storage system, which has a limited scalability. Using distributed system may improve the IO performance greatly. Another interesting issue for future work is to estimate the number of processes automatically according to the size of image, since the number of processes determines the size of data partition and the parallel granularity, how to estimate the number of process by the program automatically is also our next work.

Acknowledgments: This study is supported in part through the HTRDP (863) program of China under grant (2015AA123901) and the National Science Foundation under grant No.41301431, we will greatly appreciate the anonymous reviewers for their helpful assistance and valuable suggestions.

Author Contributions: Shiyong Liu wrote the paper and performed the research design and experiments analysis. Luo Chen and Jun Li conceived the initial research ideas and help to revise the manuscript

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

References

1. Yu, L. Accelerating image pyramid on gpus. *Advanced Materials Research* **2014**, *10*, 142-149.
2. Liu, P.; Gong, J. Parallel construction of global pyramid for large remote sensing images. *Geomatics & Information Science of Wuhan University* **2016**, *41*, 117-122.
3. Wei, X.; Lu, X.; Sun, H. Fast View of Mass Remote Sensing Images Based-on Image Pyramid. *2008. ICINIS '08. First International Conference on IEEE* **2008**, 461-464.
4. Deng, X.Q. Research on service architecture and algorithms for grid spatial data. *Acta Geodaetica Et Cartographic Sinica* **2003** *32*, 361-362.
5. Adelson, E.H.; Anderson, C.H.; Bergen, J.R.; Burt, P.J.; Ogden, J.M. Pyramid methods in image processing. *Rca Engineer* **1983**.
6. Ren, Y. A comparative study of irregular pyramid matching in bag-of-bags of words model for image retrieval. *Springer International Publishing* **2014**.
7. Yan, Z.; Xu, D.; Tan, M. A fast and robust method for line detection based on image pyramid and Hough transform. *Transactions of the Institute of Measurement & Control* **2011**, *33*, 971-984.
8. Teng, Y.; Liu, F.; Wu, R. The Research of Image Detail Enhancement Algorithm with Laplacian Pyramid. *IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing* **2013**, 2205-2209.
9. Zhao, D.; Bai, Y.Q. A novel fast haze removal technique for single image using image pyramid. *Control Conference. IEEE* **2015**.
10. Li, H.; Wang, X.; Shen, H.; Yuan, Q.; Zhang, L. An efficient multi-resolution variational retinex scheme for the radiometric correction of airborne remote sensing images. *International Journal of Remote Sensing* **2016**, *37*, 1154-1172.
11. Belle L.T. Digital image acquisition and continuous zoom display from multiresolution views using heterogeneous image pyramids. *Proceedings of SPIE - The International Society for Optical Engineering* **2003**, *5009*.
12. Momeni, H.; Sadeghi, M.T.; Abutalebi, H.R. Fast face recognition using a combination of image pyramid and hierarchical clustering algorithms. *International Conference on Wireless Communications & Signal Processing* **2014**, 1-5.

13. Huang, L.; Zhang, G.; Zhou, C. The parallel segmentation algorithm based on pyramid image for high spatial resolution remote sensing image. *Proceedings of SPIE - The International Society for Optical Engineering* **2014**, *9185*.
14. Goodchild, M. F.; Huadong, G.; Alessandro, A.; Ling, B.; Kees, D. B.; Frederick, C.; et al. An efficient multi-resolution variational retinex scheme for the radiometric correction of airborne remote sensing images. *Proceedings of the National Academy of Sciences* **2012**, *109*, 88-94.
15. Huang, C.M.; Liu, Q.; Li, Y.X. Gpu-based image pyramid transform algorithm. *Key Engineering Materials* **2012**, *500*, 422-427.
16. Kang, J.F.; Zhen-Hong, D.U.; Liu, R.Y.; Fang, L. Parallel image resample algorithm based on gpu for land remote sensing data management. *Journal of Zhejiang University* **2011**, *38*, 695-700.
17. Cheng, C.Q.; Zhang, E.D.; Wan, Y.W. Research on remote sensing image subdivision pyramid. *Geography and Geo-Information Science* **2010**, *26*, 19-23.
18. Yang, J.; Zhang, J. Parallel performance of typical algorithms in remote sensing-based mapping on a multi-core computer. *Key Engineering Materials* **2015**, *81*, 373-385.
19. Yang, J.; Zhang, J.; Huang, G. A parallel computing paradigm for pan-sharpening algorithms of remotely sensed images on a multi-core computer. *Remote Sensing* **2012**, *6*, 6039-6063.
20. 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 Transactions on Parallel & Distributed Systems* **2015**, *26*, 1497-1508.
21. Chu, B.; Jiang, D.L.; Cheng, B. Large scale mosaic using parallel computing for remote sensed images. *Applied Mechanics & Materials* **2014**, *556-562*, 4746-4749.
22. Zheng, G. Research and implementation of fast generation of massive remote sensing image in Pyramid. *East China Normal University* **2012**.
23. Yi, L. Parallel batch-building remote sensing images tile pyramid with mapreduce. *Geomatics & Information Science of Wuhan University* **2013**, *38*, 278-282.
24. He, G.J.; Xiong, W.; Chen, L.; Wu. Q.Y; Jing, N. An mpi-based parallel pyramid building algorithm for large-scale rs image. *Journal of Geo-Information Science* **2015**, *17*, 515-522.
25. ArcGIS, E.S.R.I., 2013. "10.1 Desktop help. ESRI, Redlands, CA. ArcGIS, 10".
26. Adobe Developers Association., 1992. "26.TIFF Revision 6.0. Mountain View, Cal.: Adobe Systems".
27. Qin, C.Z.; Zhan, L.J.; Zhu, A. How to apply the Geospatial Data Abstraction Library (GDAL) properly to parallel geospatial raster I/O? *Transactions in GIS* **2014**, *18*, 950-957.
28. Liu, O. Parallel access methods for geographic raster data. *Computer Science* **2012**.
29. orwal, S.; Katiyar, S.K. Performance evaluation of various resampling techniques on IRS imagery. *2014 Seventh International Conference on Contemporary Computing (IC3), IEEE* **2014**, 489-494.
30. Parker J.A.; Kenyon, R.V.; Troxel, D. Comparison of interpolating methods for image resampling. *IEEE Transactions on Medical Imaging* **1983**, *2*, 31-39.

© 2016 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 (http://creativecommons.org/licenses/by/4.0/).