Sensors and Materials, Vol. 35, No. 1 (2023) 333–346
MYU Tokyo

S & M 3171

Optimization Strategy for Building Tile Pyramid of Massive Remote Sensing Images

Xiaoli Liu[1*] and Wei Sun[2]

[1]Chinese Academy of Surveying and Mapping, 28 Lianhuachi West Rd, Haidian District, Beijing 100830, China
[2]Piesat Information Technology Co., Ltd., Zone A, Yiyuan Cultural and Creative Industry Base, Xingshikou Road, Haidian District, Beijing 100195, China

(Received October 31, 2022; accepted January 11, 2023)

Keywords: tile pyramid, massive remote sensing images, multithreading, parallelism

The efficient building of tile pyramids is the basis for achieving efficient visualization of massive remote sensing images and network publishing. With the increasing number of images and amount of data from images, the existing method leads to excessive overall time consumption and low efficiency of the building process due to the poor quality of building algorithms and parallel strategies. In this study, an optimization strategy for building a tile pyramid of massive remote sensing images was proposed. This method comprehensively used the hierarchical blank tiles of remote sensing images and the strategy of multithreading parallel pyramid construction to reorganize the existing tile pyramid construction process. The experimental results showed that the method proposed in this study was less time-consuming and had an obviously improved overall efficiency compared with that of the existing method of building a tile pyramid using massive remote sensing images. Therefore, this method is more advantageous to use than previous methods.

1. Introduction

Massive remote sensing images, especially with the characteristics of continuous time images, are remote sensing image data obtained from the same remote sensing platform and arranged in the order of their acquisition time and acquisition range; they are also called massive time-series images. Building a tile pyramid is the basis for achieving efficient visualization and network release of massive time-series images. The essence of the tile pyramid is to realize layered and blocked processing of remote sensing images within the scope of the viewpoint that is considered, to form tiles with different resolution ratios, and to construct the tile pyramid through resampling of the original remote sensing images to improve the response speed of image browsing and zooming.(1) Currently, with the increasing amount of remote sensing data and the increasing demands of network distribution, the development of methods to enable the rapid building of a tile pyramid for massive time-series images has become an urgent problem to be solved in the networked application of remote sensing images. In recent years, scholars worldwide have conducted extensive studies on the building of tile pyramids for time series

*Corresponding author: e-mail: 83391860@qq.com
https://doi.org/10.18494/SAM4228

ISSN 0914-4935

images. According to processing methods for massive time-series images, the building of a tile pyramid can be done in one of the three ways: building based on image fusion,[2–6] building based on a mosaic dataset, and building based on map plotting.[7–11] Among these methods, building based on map plotting cannot be accomplished in a normal environment owing to its reliance on a rendering engine for map plotting and its extensive demands on the computing environment. Therefore, currently, the most commonly used methods for tile pyramid building are those based on image fusion and those based on a mosaic dataset.

The building method based on image fusion usually performs image fusion and splicing according to the time series and then concurrently conducts tile cutting and pyramid building of the spliced large images. However, this method takes a long time to splice images and is not suitable for building a tile pyramid for massive time-series images with large volumes of data. The building method based on a mosaic dataset primarily uses the built-in image data model of ArcGIS to construct the tile pyramid, but this method requires manually inputting the massive time-series images. In addition, the two highlighted methods fail to realize the optimum configuration between the parallel use of the multiprocess and parallel strategy and the operational environment; thus, they have a lower overall efficiency in building a tile pyramid. Therefore, in this study, we proposed an optimization strategy for building a tile pyramid of massive remote sensing images without requiring image splicing, that conducts tile cutting according to the characteristics of the different forms of tiles in the overlapping areas of images, and that designs an optimization strategy for multithreading parallel processing to achieve the rapid building of a tile pyramid for massive time-series images.

2. Related Work

2.1 Existing methods for building a tile pyramid

2.1.1 Building method based on image fusion

The process of building a tile pyramid for massive time-series images based on the image fusion method usually includes two basic steps: image splicing and tile cutting. Image splicing mainly includes registration and fusion.[12] Although a large number of studies are focused on rapid image splicing,[13–15] the splicing process requires a large number of computations and a large amount of time owing to a large number of images and a significant amount of image overlapping.[11] Therefore, to improve the efficiency of building a tile pyramid, many studies focus on the optimization of the parallel strategy of tile cutting. Liu *et al.*[5] divided pyramid building into processes by adopting the secondary data processing strategy and achieved parallel tile building and outputting. He *et al.*[6] proposed the method of parallel pyramid building based on a message passing interface and set the parallel resampling of different particle sizes for different levels of tiles to reduce the data reading frequency. Liu *et al.*[2] proposed a parallel synthesis method for low-level tiles based on high-level tiles. The above methods are mainly based on parallel thread processing in the layered building of a pyramid to improve the building efficiency of the tile pyramid.

2.1.2 Building method based on a mosaic dataset

A mosaic dataset is a type of grid data model introduced by ArcGIS10, which is used to manage a set of grid datasets that are stored in the form of a directory and viewed in the form of a mosaic image. A mosaic dataset has the same advantages as the other two types of grid data models of ArcGIS (the grid dataset and grid directory). A mosaic dataset can quickly browse the results of dynamic mosaic images, avoid the large image splicing workload, and independently retain the personality characteristics of each image. Therefore, it is more suitable for building a massive time-series image dataset and tile pyramid. Kuz'menko *et al*.[8] used a mosaic dataset to build a multilayered structure using landscape data, while Yang *et al*.[10] used a mosaic dataset to construct a pyramid for an orthoimage dataset and data storage. These results established that this method can be used for the management and network release of large-scale image data.

2.2 Shortcomings of existing methods

Among the existing methods for building a tile pyramid for massive time-series images, the method based on image fusion is an easy one to implement. Additionally, the method based on a mosaic dataset was provided with a software tool that can meet the efficiency requirements for building a tile pyramid when the total number of images involved is small. However, dozens and even thousands of massive time-series images are generated, and the data volume can reach the level of hundreds of gigabytes ("GB" for short) and even terabytes ("TB" for short). In practical applications, the existing methods have the following limits or shortcomings.

First, massive time-series images are arranged in order of collection time and range. In general, approximately 30% of the images overlap at the boundary edges of two adjacent images. The existing method based on image splicing must carry out registration, fusion, and other operations of the overlapped multiple series of images to splice them into one complete image; then, tile cutting and building must be done. Because extraction of the registration characteristic point, image resampling, and other steps require a large number of calculations during image splicing, and because the overlapping areas are constantly accumulated as the number of images increases, these processes must be repeated and become time-consuming.

Second, to build a tile pyramid, most of the existing methods use the multiprocess parallel mode. The corresponding number of processes is determined according to the amount of parallelism, and then the cutting and outputting operations are carried out in these processes. As the creation, switching, and revocation of processes themselves will require a large calculation volume, and because balancing the tasks assigned to different processes is difficult, the processes cannot be completed simultaneously, and it becomes difficult to completely accept the advantages of parallelism. In addition, tile cutting and tile outputting are usually not matched with each other in terms of speed. The cutting speed mainly depends upon CPU performance, while the outputting speed mainly depends on the capacity of the memory and disk. As the CPU calculation speed is higher than that of the memory and because the internal calculation ability of each process is difficult to distribute equally, it is very difficult to realize the optimum utilization of the overall performance of the operating environment.

3. Method

In view of the problems described, in this study, we propose an optimization strategy for building a tile pyramid of massive remote sensing images. The core of the method includes three parts: (1) implementing tile cutting of the nonoverlapping area of massive time-series images using the bottom-up method according to the time series, (2) replacing image splicing with dst_ over tile fusion in the overlapping area of massive time-series images to cut the tiles at the same position in the overlapping areas, and (3) replacing multiprocess parallelism with multithreading parallelism. This process takes full consideration of the computational burden of tile cutting and tile outputting and optimizes the parallel strategy for building a tile pyramid.

3.1 Tile cutting in the nonoverlapping area of massive time-series images

In accordance with the coverage of the original image and preset tile size, the line and row numbers of all tiles corresponding to the image can be calculated with a formula detailed in Sect. 2.4 to obtain positional information for all tiles corresponding to the massive time-series images. Therefore, all tiles can only be obtained by cutting in regard to their position without image splicing. However, the nonoverlapping area of images contains only one tile due to overlapping among the images, while the overlapping area has many tiles that have the same line and row numbers (called "tiles at the same position" for short). Therefore, it is necessary to conduct tile cutting in these two cases.

For the nonoverlapping part of the massive time-series images, in this study, we adopt the more mature bottom-up method[2] for tile cutting. The method first carries out rendering, sampling, and parallel cutting of the bottommost tiles, which have similar tile levels and original image resolutions. The method then performs these operations on the other levels of tiles, starting directly from the bottommost tile and going upward layer by layer to reach the adjacent upper layer of tiles using the "Four in One" method. During parallel cutting, in this study, we propose to use a quantitative calculation method based on large tile cutting found in the literature.[16] The method is as follows.

One cut can directly render one large tile that has a coverage of *i*j* (line*row) pieces of tiles and then cut it to obtain *i*j* pieces of tiles to avoid the time consumed by directly implementing *i*j* tile rendering operations. To realize the maximum utilization of the running environment, the values of i and j can be calculated with the following equations:

$$i * j < M / \left(m\_thread + m\_tile\_original + m\_tile\_outpout \right), \tag{1}$$

$$i / j = ceil\left(width / height \right). \tag{2}$$

In Eq. (1), M is the memory size consumed by running the parallel program, *m_thread* is the memory space occupied by one processing program, *m_tile_original* is the memory space consumed to render one tile, and *m_tile_output* is the memory space necessary for storing one

tile. In general, the total memory space for the rendering, storing, and programming of *i***j* pieces of tiles shall not exceed the memory space to be consumed by running the parallel program.

In Eq. (2), *ceil* is the round up operation, *width* is the width of the original image, and *height* is the height of the original image, meaning that a large tile shape is consistent with the original image shape as far as possible; this step reduces the number of transparent tiles in the large tiles as much as possible.

3.2 Tile cutting in the overlapping area of massive time-series images

For the overlapping parts of the massive time-series images, the concept of tile fusion is adopted in this study for tile cutting. As shown in Fig. 1(a), when images 1 and 2 are successively obtained adjacent images, the line and row numbers of the tiles in the coverage area can be obtained on the basis of existing parameters, as shown in Fig. 1(b). Fifteen pairs of tiles at the same position will be generated in the overlapping area, as shown in Figs. 1(c) and 1(d). The dst_over[17] method proposed by the Worldwide Web Consortium (W3C) is used for the fusion of these tiles at the same position.

For completely overlapped tiles at the same position, such as tile Nos. 4, 11, 18, 25, and 32 at the same position, the dst_over tile fusion operation is carried out, and the result was that tile No. 2 of the new image was used to replace the tile of the original No. 1 image.

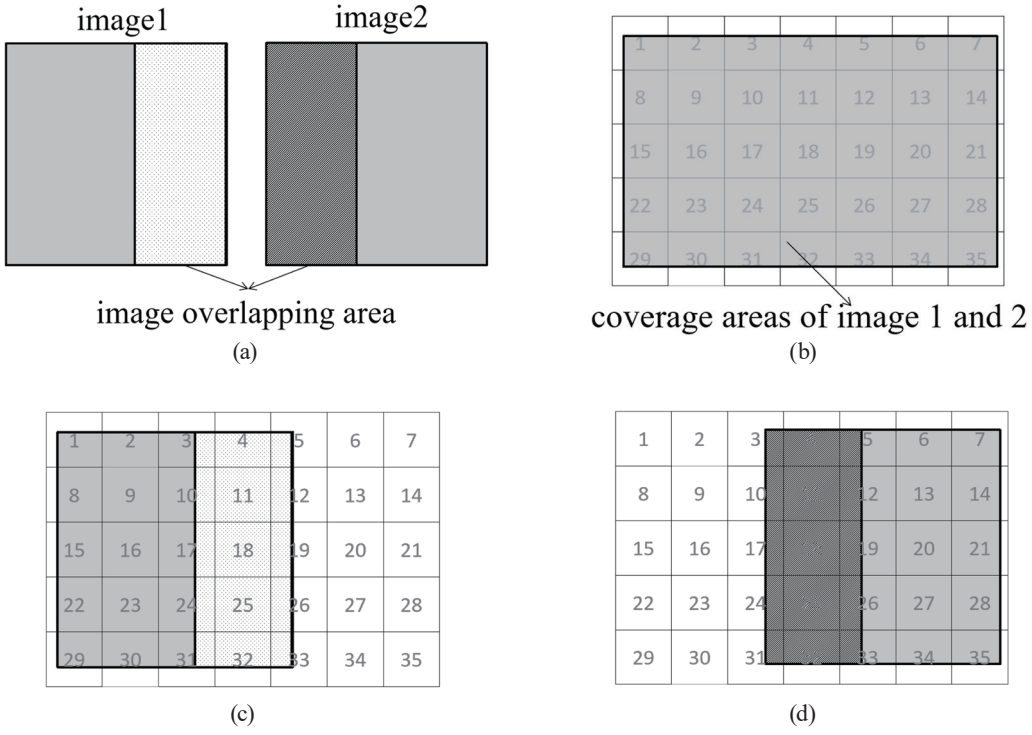


Fig. 1. Schematic diagram for tile cutting in the overlapping area. (a) Overlapping image of two adjacent landscapes. (b) Line and row numbers obtained through the calculation of two images. (c) Coverage tile of image 1. (d) Coverage tile of image 2.

For partially overlapping tiles at the same position, which are the tiles located at the edge boundary location of images, such as tile Nos. 3, 5, 10, 12, 17, 19, 24, 26, 31, and 33, the fusion process of tile No. 10 at the same position in Figs. 1(c) and 1(d) was taken as an example, as shown in Fig. 2. Figure 2(a) corresponds to tile No. 10 obtained from the cutting of image 1. The left side of the figure is a pixel representation of the nonoverlapping area of the image, while the right side is a pixel representation of the overlapping area of image 1. Figure 2(b) corresponds to tile No. 10, which is obtained from the cutting of image 2. The left side shows transparent pixels, while the right side shows the pixels of the overlapping area. In the dst_over tile fusion operation, the area of transparent pixels still uses the pixel value of the tile shown in Fig. 2(a), and the pixels of the overlapping area use the pixel value of the tile shown in Fig. 2(b) to obtain the resulting tile, shown in Fig. 2(c). Through tile fusion, the tile of the overlapping area can be obtained by cutting without image splicing.

3.3 Optimization of the multithreading parallel strategy

To make full use of the computing resources of the running environment, in this study, we replace the multiprocess with multithreading, which has lower resource consumption and fast switching for implementing the parallel strategy for building a tile pyramid.[18–20] Over the whole course of implementation of the strategy, only one process is enabled to conduct parallel processing of the threads for the tile cutting and tile outputting operations. That is, the cut operation is controlled using the thread pool, while the output operation is controlled by the output queue. Additionally, the hardware parameters of the existing running environment are used to calculate the state of the balance points between the two operations and to improve the building efficiency of the tile pyramid on the whole; the processing flow is shown in Fig. 3.

All of the threads in the thread pool have three states: sleeping, activating, and dead. In addition, the control command of the thread pool is responsible for handling the allocation of each thread in the thread pool.

For the cut processing of each tile in the cutting operation, a new independent thread is created and added into the cut thread pool (process 2 in Fig. 3). In accordance with the large tile cut form described in Sect. 2.1, the capacity of the thread pool is set to $i*j$, which is the number of tiles covered by one large tile. The new threads created are first in a sleeping state, and the control command accordingly activates n threads for the cutting operation according to the set

(a) (b) (c)

Fig. 2. Schematic diagram for the fusion of tiles at the same position. (a) Tile No. 10 from image 1. (b) Tile No. 10 from image 2. (c) Resulting tile after fusion.

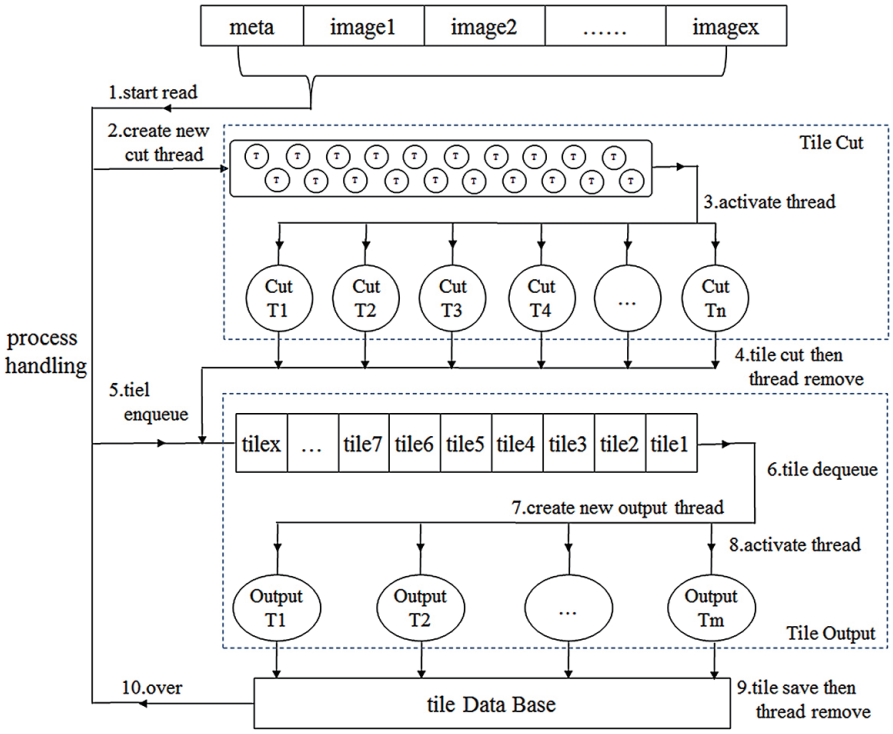


Fig. 3. Multithreading parallel processing flow for the methods in this study.

number (n) of parallel cut threads (process 3 in Fig. 3). When the operations of certain threads are completed, the control command sets this thread in the dead state and removes it from the thread pool (process 4 in Fig. 3). In this way, the cut thread always continues to dynamically update the process to make the total number of threads in the cut thread pool always larger than the number of threads in the parallel process until the completion of the cutting operation to ensure that the entire cut course can make full use of the CPU calculation capacity of the running environment.

In the output operation, each piece of tile that has been properly cut enters the output queue in order (process 5 in Fig. 3), and the total length of the output queue (total number of storage tiles) is set to the number of tiles ($n*i*j$) in the parallel cut processing, which is the total number of tiles resulting from one cutting operation of the cut thread pool. The enqueue and dequeue sequences of tiles follow the principle of "first in first out" (process 6 in Fig. 3). Similarly, the control command activates the output thread according to the set number of parallel output threads and stores the tile output in the database (processes 7, 8, and 9 in Fig. 3).

The sum of the number of parallel cut threads (n) and the number of parallel output threads (m) is equivalent to the number of CPU logic cores in the running environment, and the specific values of n and m are related to the resolution of the original image and the calculation capacity of the specific running environment, respectively. This cut operation relies more on the CPU performance and $n > m$ under normal conditions. Prior to the large-scale official building of a tile pyramid for massive time-series images, some images may be selected in advance for experimentation and a reasonable distribution of the n and m values.

3.4 Algorithm process and steps

In accordance with the building method, the established building algorithm for a tile pyramid for massive time-series images includes the following steps.

Step 1: Calculate the number of layers (*Lmax*) at the lowest level of the tile pyramid corresponding to the massive time-series image. In accordance with the spatial reference information in the image metadata, the worldwide scope [*Xmin, Ymin, Xmax, and Ymax*] of the constructed tile pyramid is determined. In accordance with the tile size, Eq. (3) is used to perform the cycle calculation of the ground resolution *L_resolution* corresponding to each layer of the tile pyramid.

$$L\_resolution = \left(Xmax - Xmin\right) / \left(tile\_size * 2^{level}\right), \quad level = 0,1,2,\ldots,n \tag{3}$$

In the course of the cycle calculation, each calculated L_resolution value is compared with the image resolution. If *L_resolution<image resolution*, the corresponding level grade of this *L_resolution* is the number of layers *(Lmax*) at the lowest level of the tile pyramid.

Step 2: Calculate the line and row numbers of tiles at the lowest layer (*Lmax*) of the image of the first landscape. In accordance with the extent of coverage of the image and corresponding ground resolution (*Lmax_resolution*) at the lowest layer *Lmax*, Eq. (4) is used to calculate the line and row numbers that occupy this image.

$$\begin{aligned}
Tile_Xmin &= floor\left(\left(extent_Xmin\text{-}Xmin\right) / \left(Lmax_resolution * tile_size\right)\right);\\
Tile_X\max &= ceil\left(\left(extent_X\max\text{-}Xmin\right) / \left(Lmax_resolution * tile_size\right)\right);\\
Tile_Ymin &= floor\left(\left(Ymin - extent_Ymin\right) / \left(Lmax_resolution * tile_size\right)\right);\\
Tile_Y\max &= ceil\left(\left(Y\max - extent_Y\min\right) / \left(Lmax_resolution * tile_size\right)\right).
\end{aligned} \tag{4}$$

Step 3: Implement tile rendering and cutting of tiles at the lowest layer (*Lmax*). Use the method described in Sect. 3.1 for tile cutting and then output it to the output queue. To reduce the time consumption of the redundant tile outputting and storage, the output is given if the tiles are transparent tiles after cutting.

Step 4: Implement the fusion of tiles in the lowest layer (*Lmax*). Repeat steps 2 and 3 for the tiles of the second landscape and judge whether there are tiles at the same position in the output queue when tile outputting is implemented in procedure 3. If there are tiles at the same position, use the method described in Sect. 3.2 to fuse the tiles at the same position. Execute this operation until the cutting and outputting of the tiles at the lowest layer (*Lmax*) of all images is completed.

Step 5: Generate tiles for the other layers. After completing the outputting of the tiles at the lowest layer, the tiles of the other layers are formed through tile merging to complete the building of the tile pyramid.

4. Experiment and Analysis

4.1 Experimental data and environment

The experimental data are massive time-series image data from the Gaofen-2 satellite from a specific region of China's Yangtze River Delta, with the coverage extent shown in Fig. 4. The duration of data collection was from April 30, 2017, to December 31, 2017. There are 256 landscapes for which the image data size of each landscape was within 3.7–5.3 GB, and the total data size was 1.3 TB. All of these data were highly representative. The experimental data have an image resolution of 0.81 m and a spatial reference of WGS84. In accordance with Eq. (3), the layer number at the lowest level of the tile pyramid corresponding to this massive time-series image was calculated to be 18, and the layer number of the topmost level is 10. The sampling method for tile generation is the bilinear sampling method. In accordance with the massive time-series image parameters and Eqs. (1) and (2), the number of large tiles rendered at a time is calculated to be 8*8. The building method based on image fusion that is represented in Ref. 2 requires massive time-series image splicing in advance and is realized by using Erdas in this study. According to Sect. 3.3, the optimal parallel thread allocation scheme was determined using images of 4 landscapes in advance, of which the number of parallel cut threads was 6 and the number of parallel output threads was 2 based on the results of calculations. The method in Ref. 2 and the ArcGIS mosaic dataset method both use the multiprocess parallel mode, and the number of parallel processes set for these two methods was 8, which was equivalent to the total number of parallel threads in this study.

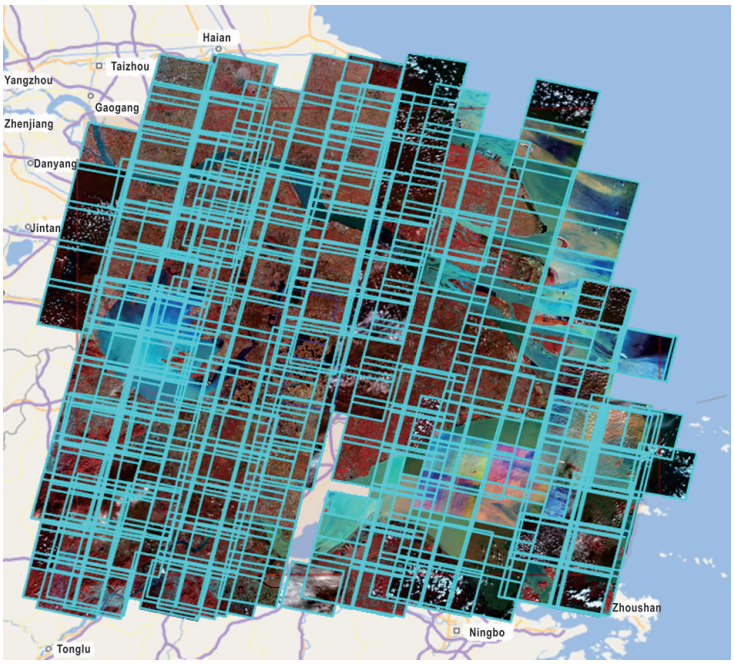


Fig. 4. (Color online) Overview of the scope of the experimental area and experimental data.

4.2 Comparison of the efficiency of methods

The method proposed in this study, the method from Ref. 2, and the ArcGIS mosaic dataset method were used in the building experiment for the tile pyramid for massive time-series images in the same running environment. The data are shown in Tables 1–3, and the total time consumption results are shown in Fig. 5.

It was observed that the average time consumed by the method proposed in this study was 13.62 times less than that of the method from Ref. 2 and 4.84 times less than that of the ArcGIS mosaic dataset method. Among these methods, the time consumption of image splicing in the method from Ref. 2 was 67.3% of the total time consumption, and with the increase in the

Table 1
Time consumed for building the tile pyramid for massive time-series images using the method proposed in this paper. (unit: s)

Number of landscapes for massive time-series images	Time consumed for tile cutting at lowest layer	Time consumed for merging and producing tile pyramid	Total time consumed
4	139.363	97.756	237.119
16	1078.837	353.124	1431.961
32	2406.361	597.643	3004.004
64	5158.412	853.647	6012.059
128	10507.174	1533.871	12041.045
256	23100.998	3145.524	26246.522

Table 2
Time consumed for building the tile pyramid for massive time-series images using the method in Ref. 2. (unit: s)

Number of landscapes for massive time-series images	Image splicing	Formation of pyramid	Total time (s)
4	822.77	920.79	1743.56
16	3820.79	5167.37	8988.16
32	13611.88	11297.03	24908.91
64	31446.53	26600.99	58047.52
128	104829.7	53978.32	158808.02
256	294427.72	120201.98	414629.7

Table 3
Time consumed for building the tile pyramid for massive time-series images using the ArcGIS method. (unit: s)

Number of landscapes for massive time-series images	Time consumed for building mosaic dataset	Time consumed for adding image data	Time consumed for building external enclosing frame	Time consumed for building image pyramid	Time consumed for formation of tile pyramid	Total time (s)
4	0.59	1.34	1.57	12.89	1157	1173.39
16	0.61	1.36	1.47	13.56	7043	7060
32	0.68	3.02	26	505	14215	14749.7
64	0.66	12.98	64	984	28360	29421.64
128	0.58	23.3	134	2001	56398	58556.88
256	0.64	52.06	277	4366	121520	126215.7

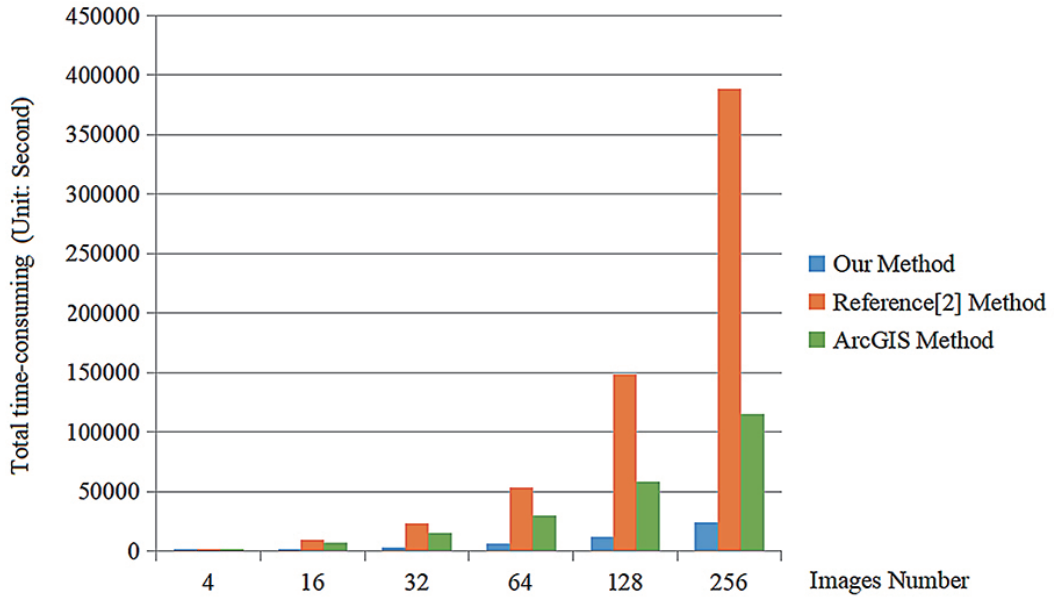


Fig. 5. (Color online) Comparison of the total time consumption of tile pyramid building for massive time-series images by different methods.

number of landscapes in the images, the time consumed for splicing exponentially increased, and the total time consumption was high. The time taken by the method from Ref. 2 to construct the tile pyramid after image splicing was 1.05 times faster than that of the ArcGIS mosaic dataset method due to the partial optimization of its tile cutting method. In addition, with the increase in the number of image scenes, the total time-consumption ratio between the method proposed in this study and the ArcGIS mosaic dataset method was reduced from 4.95 to 4.81. This reduction occurred because the corresponding tile fusion time was slightly increased due to the growth of the number of tiles at the same position cut using the method proposed in this study after the number of landscapes was increased.

4.3 Discussion of the CPU utilization ratio

To further verify that the parallel strategy proposed in this study makes better use of the CPU calculation ability to build a pyramid for massive time-series images, the method proposed in this study was compared with the method from Ref. 2 and ArcGIS mosaic dataset method in terms of the CPU utilization ratio. The time used in the statistics was 2 min at intervals of 1 s; the results are shown in Fig. 6. It was observed that the method proposed in this study basically maintained an average CPU utilization ratio of approximately 97%, while the method from Ref. 2 and the ArcGIS mosaic dataset method only maintained an average CPU utilization ratio of approximately 70–80%. Additionally, as the CPU utilization characteristics of tile cutting and tile outputting were not considered, the CPU utilization ratio exhibited a steep drop after a certain period.

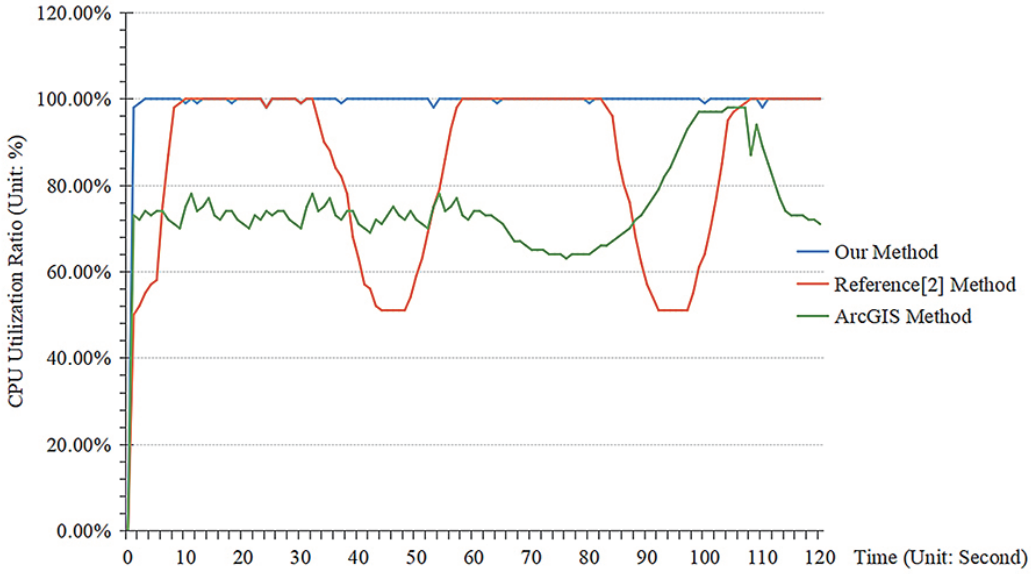


Fig. 6. (Color online) Comparison of the CPU utilization ratio.* (Note*: To accurately reflect the consumption of the CPU in the building process of the tile pyramid, the method from Ref. 2, and the ArcGIS mosaic dataset method, we only selected the corresponding CPU consumption of the tile pyramid building without calculating the CPU consumption of the image splicing, image adding, and other processes.)

5. Conclusions

The existing methods to build a tile pyramid for massive time-series images result in a long overall time consumption and low efficiency due to poor quality of building algorithms and parallel strategies when the number of images and amount of data from images are large. For this reason, in this study, we proposed an optimization strategy for building a tile pyramid from massive images. We implemented tile cutting according to the characteristics of different forms of tiles in the areas of image overlap, constructed a pyramid without image splicing, and adopted multithreading to optimize the parallel strategy in building the tile pyramid to realize the efficient and rapid building of the tile pyramid for massive time-series images. The main conclusions obtained through actual data verification of the massive time-series images are as follows.

(1) In terms of the efficiency of the methods, the method proposed in this study, the method based on image fusion represented by Ref. 2, and the ArcGIS mosaic dataset method were used to conduct an experiment in the building of a tile pyramid for massive time-series images in the same running environment. The method proposed in this study had the shortest time consumption; it is 13.62 times faster than the method from Ref. 2, and 4.84 times faster than the ArcGIS mosaic dataset method.

(2) In terms of CPU utilization, the method proposed in this study maintains a relatively stable average CPU utilization ratio of approximately 97%. The method from Ref. 2 and the ArcGIS mosaic dataset method have average CPU utilization rates of only approximately 70–80%.

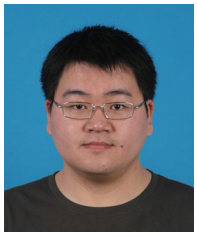
The method proposed in this study was more suitable for operation processing in a single machine environment. When the calculation environment is a distributed multinode collaboration, it is also necessary to carry out parallel strategy transformation of the corresponding environment, which is a future direction for this research.

References

1 I. Viola, A. Kanitsar, and M. E. Groller: Proc. 2003 IEEE Visualization (IEEE, 2003) 309–316.
2 S. Liu, Q. Wu, C. Luo, J. Li, and J. Ning: Geomatics World **22** (2015) 51 (in Chinese).
3 L. Yi: Geomat. Inf. Sci. Wuhan Univ. **38** (2013) 278 (in Chinese). https://doi.org/10.13203/j.whugis2013.03.006
4 Y. Liu, L. Chen, W. Xiong, L. Liu, and D. Yang: Proc. 2012 IEEE Int. Conf. Geoinformatics (IEEE, 2012) 1–7.
5 P. Liu and J. Gong: Geomatics Inf. Sci. Wuhan Univ. **41** (2016) 117 (in Chinese). https://doi.org/10.13203/j.whugis20130718
6 G. He, W. Xiong, L. Chen, Q. Wu, and N. Jing: J. Geo-Inf. Sci. **17** (2015) 515 (in Chinese). https://doi.org/10.3724/SP.J.1047.2015.00515
7 ArcGIS. What is a mosaic dataset? http://desktop.arcgis.com/en/arcmap/10.3/manage-data/raster-and-images/what-is-a-mosaic-dataset.htm (accessed November 2022).
8 E. I. Kuz'menko, A. A. Frolov, and A. V. Silaev. Geogr. Nat. Resour. **39** (2018) 175. https://doi.org/10.1134/S1875372818020117
9 J. Dong, Y. Huang, and P. Dong: Infrared **33** (2012) 30 (in Chinese).
10 M. Yang, J. Liu, X. Li, and Y. Zhang: Geomatics World **1** (2014) 33 (in Chinese).
11 N. Guo, W. Xiong, Q. Wu, and N. Jing: Adv. Electr. Comput. Eng. **16** (2016) 3. https://doi.org/10.4316/AECE.2016.04001
12 M. Brown and D. G. Lowe: Int. J. Comput. Vision **74** (2007) 59. https://doi.org/10.1007/s11263-006-0002-3
13 Y. Yang, Y. Gao, H. Li, and Y. Han: AProc. 2011 Int. Symp. Image and Data Fusion (IEEE, 2011) 1-4.
14 X. Xie, X. Yin, Q. Liu, F. Hu, T. Cai, J. Nan, and H. Xiong: J. Ambient Intell. Hum. Comput. **6** (2015) 835. https://doi.org/10.1007/s12652-015-0319-2
15 J. Wang, J. Fang, X. Liu, D. Zhao, and Q. Xiao: Int. J. Remote Sens. **35** (2014) 5959. https://doi.org/10.1080/2150704X.2014.943320
16 X. Liu, Y. Zhang, J. Yang, and L. Hao: Sci. Surv. Mapp. **41** (2016) 144 (in Chinese). https://doi.org/10.16251/j.cnki.1009-2307.2016.01.027
17 Northway C. Understand Compositing and Color extensions in SVG 1.2 in 30 minutes, http://www.svgopen.org/2005/papers/abstractsvgopen (accessed November 2022).
18 A. C. Sodan, J. Machina, A. Deshmeh. K. Macnaughton, and B. Esbaugh: Computer **43** (2010) 24. https://doi.org/10.1109/MC.2010.75
19 L. Zhou, C. Jiao, and J. Lan: Microcomput. Inf. **17** (2005) 118 (in Chinese).
20 C. Qin, L. Zhan, and A. Zhu: How to Apply the Geospatial Data Abstraction Library (GDAL) Properly to Parallel Geospatial Raster I/O? Transactions in GIS **18** (2015) 950. https://doi.org/10.1111/tgis.12068

About the Authors

Xiaoli Liu received her Ph.D. degree from Shandong University of Science and Technology, Qingdao, China. She is currently an associate researcher at the Chinese Academy of Surveying and Mapping, Beijing, China. Her research interest is mainly in GIS software architecture design and development, and she has published several research papers in scholarly journals in this area.



Wei Sun received his Ph.D. degree from Shandong University of Science and Technology, Qingdao, China. He is currently a senior engineer at Piesat Information Technology Co., Ltd., Beijing, China. His research interest is mainly in GIS software architecture and parallel algorithms. He has published several research papers in scholarly journals in this research area and has participated in several conferences.