# An Asynchronous Parallelized and Scalable Image Resampling Algorithm with Parallel I/O

Yan Ma[1,2,3], Lingjun Zhao[1], and Dingsheng Liu[1]

[1] Center for Earth Observation and Digital Earth, Chinese Academy of Sciences(CAS)
No. 45 BeiSanHuanXi Road, P.O. Box 2434, Beijing, 100086, China
[2] Institute of Electronics, CAS. No. 19 Beisihuan Xilu, Beijing 100190, China
[3] Graduate University of Chinese Academy of Sciences
{yanma,ljzhao,dsliu}@ceode.ac.cn

**Abstract.** Image resampling which is frequently used in remote sensing processing procedure is a time-consuming task. Parallel computing is an effective way to speed up. However, recent parallel image resampling algorithms with massive time-consuming global processes like I/O, always lead to low efficiency and non-linear speedup ratio, especially when the amount of computing nodes increases to a certain extent. And what's more, the various georeferencing related to different processing applications caused a real problem of code reuse. To solve these problems, Parallel Image Resampling Algorithm with Parallel I/O (PIRA-PIO) which is an asynchronous parallelized image resampling algorithm with parallel I/O is proposed in this paper. Parallel I/O of parallel file system and asynchronous parallelization which using I/O hidden policy to sufficiently overlap the computing time with I/O time are used in PIRA-PIO for performance enhancement. In addition, the design of reusable code like design pattern will be used for the improving of flexibility in different remote sensing image processing applications. Through experimental and comparative analysis, its outstanding parallel efficiency and perfect linear speedup is showed in this paper.

## 1 Introduction

Image resampling algorithm deals with geometric warping and transformation between two images. It plays an important role in remote sensing image processing; most of remote sensing image preprocessing procedures involve image resampling, such as Geometric Correction, Image Fusion and Image Mosaic.

Image resampling is a compute-intensive and time-consuming task. Due to the increasing spatial and spectrum resolution, the unprecedented image scales pose many computational challenges. Parallel computing with scalable cluster is an effective way to improve processing performance. Up till now, many studies on parallelization of image resampling have been probed into, including the 2D and 3D parallel image resampling algorithm on parallel machine[1], and load- balanced parallel image warping algorithm[2][3],etc. These parallel algorithms usually use Master-Slave parallel processing model. And their works mainly focus on how to reduce the computing overhead, like reducing algorithm complexity, load balancing and so on. However,

G. Allen et al. (Eds.): ICCS 2009, Part II, LNCS 5545, pp. 357–366, 2009.

there exists massive time-consuming global operation of master node in those algorithms, like serial data loading and exporting, data scattering and gathering. Moreover, with the rapidly upgrade of CPU, the performance gap between CPU and I/O is widening. As we are using a large number of CPUs (computing nodes), the I/O overhead caused by data loading and data scattering is not negligible any more, and the I/O quickly became the performance bottleneck[4].

Complete image resampling procedures commonly involve geometric mapping and interpolation. When image resampling is applied to different image processing procedures, the method of building geometric mapping varies and also the demanded auxiliary data differs, which consequently make the code reuse of this algorithm a big problem. Thus the building of different image processing applications requires developing corresponding image resampling programs which closely related to these applications, which brings another challenge to the designing and development of remote sensing image processing system.

To properly settle those issues above, PIRA-PIO an asynchronous parallelized and scalable image resampling algorithm with parallel I/O is proposed in this article. In PIRA-PIO algorithm, the data distribution and parallel I/O of parallel file system is imposed for eliminating I/O performance bottleneck, and also the I/O hidden policy which sufficiently overlaps the computing and I/O time overhead is used for further speed up. In addition, through the interface abstraction and scalability designing of algorithm with designing pattern, the image resampling program can be reused among different image processing applications.

## 2 Traditional Parallel Image Resampling Algorithm

Image resampling algorithm is complicated and time-consuming. In cluster environment, traditional parallel image resampling algorithms dealing with huge data generally adopt Master-Slave parallel processing model [7][8], their processing flow is illustrated as follows.

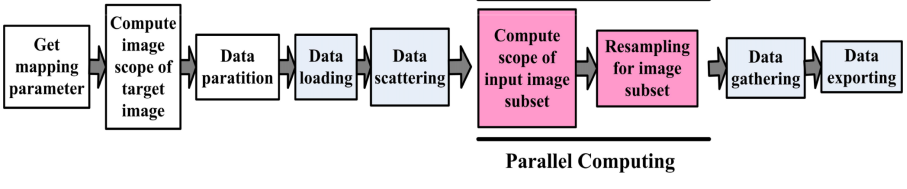



**Fig. 1.** Processing flow of traditional parallel image resampling algorithm

As illustrated above, the traditional parallel image resampling algorithms using classic parallel processing model which conduct a simple way of parallelization. The algorithm have good data locality only when the image distortion is small. Furthermore, it also has several problems. Firstly, it limits in performance scalability. As is showed in figure 1 that some processing steps are global operations which could not be parallelized, such as data loading, data scattering, data gathering and data exporting . While master node does data loading and data scattering, the computing nodes

are all in idle state waiting for the required data, only till the input image subset have already been fetched from master node can the resampling be continued. So, the overhead caused by resampling computing operation and I/O operation waiting for each other directly affects the parallel efficiency of whole algorithm. When a large amount of CPUs (computing nodes) are used, the problem result from the performance gap between I/O and CPU become even severe, and would finally brings perform bottleneck. Secondly, traditional parallel image resampling algorithms which closely related to applications in some aspects are not reusable.

## 3 PIRA-PIO Algorithm

To solve the performance problems mentioned above, we put forward PIRA-PIO algorithm an asynchronous parallelized image resampling algorithm using parallel I/O. By combining the data distribution and parallel I/O of cluster based parallel file system with the concurrent data I/O operations triggered by all computing nodes, PIRA-PIO algorithm enhances I/O performance at two layers including file system layer and algorithm layer. Moreover, based on the I/O speedup, the I/O hidden policy is also applied to overlap the I/O time with resampling computing time. Additionally, in order to improve the scalability and adaptability of this parallel algorithm, code reuse designing is carried through in algorithm building.

### 3.1 Algorithm Introduction

The main ideal of PIRA-PIO is: Master node responses for getting the scope of target image with mapping function, task partition according to the load measured by amount of efficient computation and task assignment. Each sub-task deals with a block of data block_t as is illustrated in figure 2. Each computing node who acts as slave node does three operations simultaneously: resampling for current block, prefetching next task and next data block of input image required for resampling, exporting the previous resampled data block of target output image. And each slave concurrently does data prefetching or exporting with parallel I/O supported by parallel file system PVFS2.

In PIRA-PIO algorithm, the master node maintains a task pool and schedules these sub-tasks to slave nodes according to a centralized load balance strategy which will probably leads to a load balance among slave nodes. And also instead of dividing and scattering the input image, computing nodes load data when needed through parallel I/O, which avoids the frequent data exchanges among computing nodes.

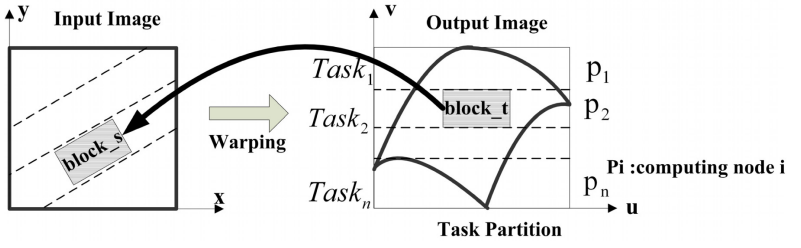



**Fig. 2.** Image mapping and data partition method of PIRA-PIO

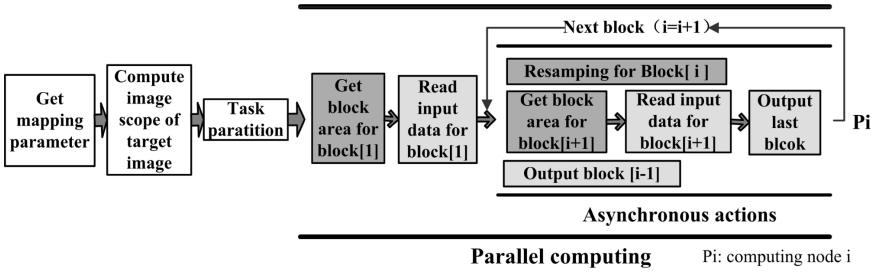



**Fig. 3.** Processing flow of PIWA-PIO

Further, from the flows in figure 3 we will see that there are no more data scattering and data gathering, and the data loading and exporting are implemented in parallel by each computing node concurrently. Thus the processing flow is shortened. And what's more, the data prefetching and asynchronous data I/O also lead to the sufficiently overlapped I/O time with resampling computing time.

### 3.2 Parallel I/O and Data Distribution

To eliminate the I/O performance, we use data distribution and parallel I/O technique in PIRA-PIO algorithm. And the remote sensing images are all stored in parallel file system PVFS2. Through PVFS2 the storage devices like disk arrays mounted to the I/O nodes in cluster are virtualized to a single file system mirror as showed in figure 4(b). The remote sensing image data in PVFS2 are partitioned into data blocks, and scattered among the disk arrays of computing nodes. The sketch map of data distribution method is showed in figure 4(a).

When implementing resampling for blocks, each computing node has to dynamically load the data in input image blocks. Due to the single file system mirror of pvfs2, all the computing nodes can access any data whenever they want. And the

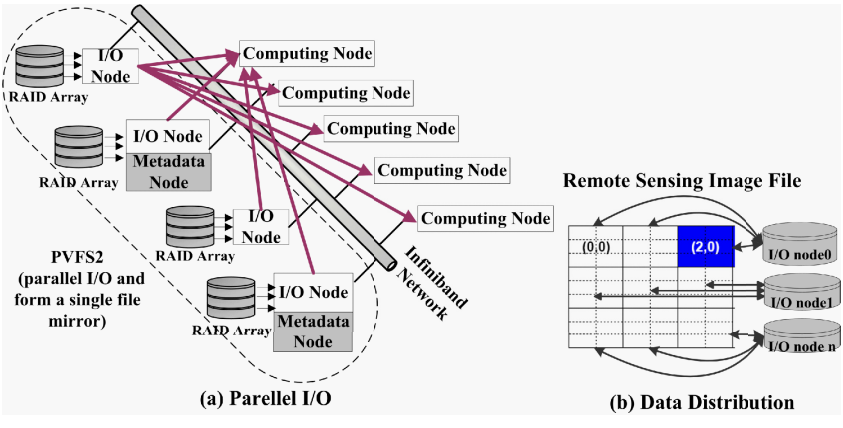



**Fig. 4.** Data distribution and parallel I/O

multi-stream concurrency technology also makes the concurrent data I/O operation possible. As a result, the computing nodes can concurrently read the data needed. While a data read operation is called, the parallel file system transforms it to several read actions directly performed on corresponding I/O nodes in parallel, and then the daemon process running on each I/O node fetches data from disk arrays in parallel. Also, when several data exporting or writing operations are simultaneously triggered by computing nodes after finishing data resampling, each computing node distributes the data directly to I/O nodes in parallel.

Consequently, PIRA-PIO algorithm not only supported parallel I/O in file system level, but also in algorithm level. And the parallel I/O provided at different system level significantly speedups I/O performance and allows a better scalability than ever.

### 3.3 I/O Hidden Strategy

Commonly, the data resampling operation has to wait while the data I/O like data loading or exporting has not yet finished, as the data used for resampling is not ready or the previous result data has not yet written back. Accordingly, even the I/O time overhead has greatly reduced by using parallel I/O, there still remains time overhead caused by the computing processing waiting for I/O. To handle the I/O waiting time issue, we put forward an I/O hidden strategy.

Data caching mechanism commonly used in the designing of storage system is an effective approach to I/O hidden. And the approach of data prefetch and asynchronous I/O used in our I/O hidden strategy is also some kind of data caching mechanism. While doing resampling for current block, it's possible for computing node to simultaneously prefetch required data for next block by calling an asynchronous I/O. Then after finishing resamping for current block, the data required for resampling next block has already prepared. Similarly, when finishes resamping for current block, the computing node would not output resampled data at once, but cache this data in buffer. Thus the computing node can start resampling for next block at once, no more wait for outputting the resampled data of current block. And that when the same time as the resamping for next block is computing, the computing can trigger an asynchronous I/O action to output the resampled data of current block that cached in buffer. As a result, data I/O operation and resampling could be implemented simultaneously, and there is no need for the resampling operation to wait for data I/O, finally the I/O could be hidden.

As is depicted in figure 5, each computing node maintains a ring buffer, which is shared among three independent threads. These three independent threads including one I/O reading thread which is responsible for getting the block area in input image for the next block and then read the data in that block area, one computing thread performs the data resampling for current block, and one I/O write thread who output the result data of previous block through pvfs2. And these three threads are putting into a pipeline as showed in following figure. When the pipeline starts, three threads work concurrently. The I/O time is mostly overlapped by computing time spent on resmapling. Finally, the I/O waiting time is almost eliminated.

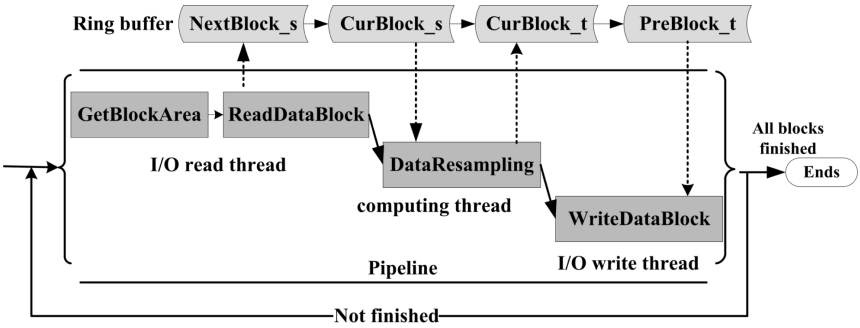



**Fig. 5.** I/O hidden strategy. 'NextBlock_s': input data for next block resampling, 'CurBlock_s' : input data for current block resampling, 'CurBlock_t': resampled data of current block, 'PreBlock_t': caching the resampled data of previous block.

### 3.4 The Code Reuse Designing

Image resampling is widely applied in most image correction processes. For example, in systematical geometric correction, a grid can be calculated by satellite orbit parameters and stance data, and then mapping relation will be obtained from the grid information, and finally follows the gray value interpolation. In ortho correction, correlative model coefficient can be calculated by satellite orbit information and ground control points, and then follows the obtaining of mapping relation with DEM, and gray value interpolation. From the traditional processing flows above, we can infer that despite of different applications, the computing flows of resampling algorithm remain almost the same. But the computing part which related to mapping relation includes obtaining methods, forms, transferring methods of it is not the same, and it is especially inconvenient for code reuse. So from the aspect of code reuse, a flexibility design based on software design pattern in algorithm implementing will be used to improve the parallel resampling algorithm's reusability in image processing system. Thereby, the programming efficiency may be improved.

When doing algorithm developing, PIRA-PIO is divided into three function modules: parallel flow control module, application computation module and gray value interpolation module. In details, parallel flow control module is in charge of realizing the parallel flow of PIRA-PIO mentioned above and making use of the correlative operation of application computation module to finish partial resampling of each computing node, grey interpolation module is responsible for calculating the appointed pixel's gray value with appointed interpolation method such as NN, BL, CC and etc, application computation module is related to applications, and it is used to compute mapping relation according to appointed criterion.

To improve the Scalability of each module, the principle of Object-Oriented design and design pattern like factory pattern and strategy pattern is used. For instance, the class diagram of application computation module is showed as Figure 6.

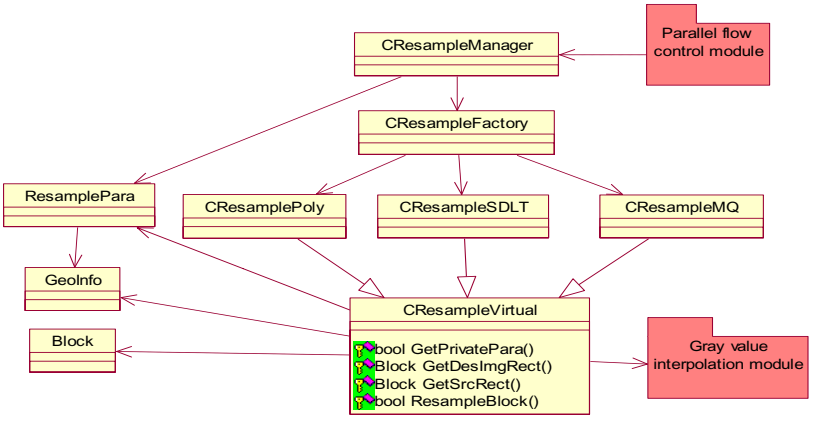



**Fig. 6.** The class diagram of application computation module

## 4 Performance Evaluation and Comparison

**(1) Performance of Traditional Parallel Resampling Algorithm**

The speedup ratio $S_p$ is:

$$S_p = \frac{T_s}{T_p} \approx \frac{2T_{I/O} + T_{process}}{2T_{I/O} + 2T_{distribue} + T_{process}/N + t_{ex}} \tag{1}$$

And the data throughput ratio R is:

$$R = \frac{M}{T} = 1/\left(\frac{2}{R_{input}} + \frac{2}{N * R_{distribution}} + \frac{1}{N * R_{process}}\right) \to R_{I/O}/2 \tag{2}$$

From the above formula we can infer that when using a large amount of computing nodes, the data throughput ratio R will be tend to $0.5R_{input}$, and the $T_{I/O}$ and $T_{distribute}$ will become the limiting factor of $S_p$.

**(2) Performance of PIRA-PIO Algorithm**

Assume that the speedup ratio of the parallel I/O with N I/O nodes is $K_n$. If each computing node assigned m sub-tasks, then the time used for read or writing a block with parallel I/O equals $T_{I/O}/(n * m * k_n)$.

Speedup ratio $S_p'$ is:

$$S_p' = \frac{T_s'}{T_p'} \approx \frac{2T_{I/O}/k_n + T_{process}}{2T_{I/O}/(n * m * k_n) + T_{process}/n} \to n \tag{3}$$

Data throughput ratio R ' is:

$$R' = \frac{M}{T_p'} \approx nR_{process} \tag{4}$$

As the $T_{I/O} / k_n * T_{process}$ less than 1, and $2T_{I/O} / T_{process} * m * k_n$ far less than 1, so when N is big enough, the speedup ratio $S_p'$ will tend to N. Consequently, an elegant linear speedup ratio can be achieved. And the data through can up to N times of the data throughput ratio of the resampling processing on each computing node.

## 5 Experiment and Analysis

The experimental performance comparison of PIRA_PIO algorithm with traditional parallel resampling algorithm will be taken on Lenovo Deepcomp 6800 cluster. Deepcomp is composed of 18 nodes, each with dual Intel(R) 3.0Ghz processors. Network capability is 1000MBps. Three bands of Beijing-1 satellite images each with size of 15000*10000 pixels are used. And in the experiment, the image resample algorithms are applied to do a fine geometric correction with Cubic Convolution resample mode, MQ mapping model and Complete Cubic Polynomial mapping. The performance comparison is carried as follows:

**(1) Comparative analysis of speedup and parallel efficiency**

As is showed in the figures below that the speedup ratio and data throughput ratio of PIRA-PIO is obviously superior to the traditional one. It is proved by the experiments that the PIRA-PIO algorithm has achieved linear speedup with the increase of computing nodes. The speedup ratio is among N-1 to N, when N computing nodes are used. But for the traditional algorithm, the speedup figure is like a curve, when the amount of computing nodes is larger than four, its speedup ratio, data throughput ratio all drop sharply.

Moreover, with the increase of computing nodes, the parallel efficiency of the traditional algorithm almost descends linearly, while the parallel efficiency of PIRA-PIO always kept better than 90%. So, we can draw the conclusion that the PIRA-PIO algorithm owns an outstanding parallel efficiency and scalability in cluster platform.

**(2) The analysis of the overhead of main processing steps**

The main time overhead of PIRA-PIO algorithm is spent on data resampling ($T_{process}$). And the $T_{io}$ which is spent on loading the first block and output the last result block is relatively very small. From the sub figure (b) we can see that with the increase of computing nodes, the $T_{process}$ linearly declines and $T_{io}$ drops too. So, when enough computing nodes are used, the $T_{io}$ can be ignored. The I/O operation almost rarely affects the performance of the whole algorithm.

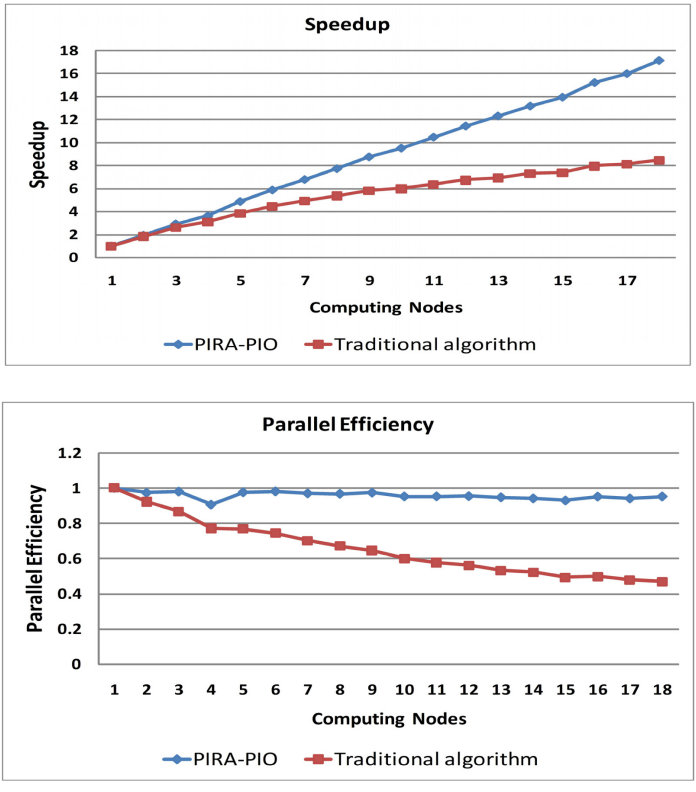
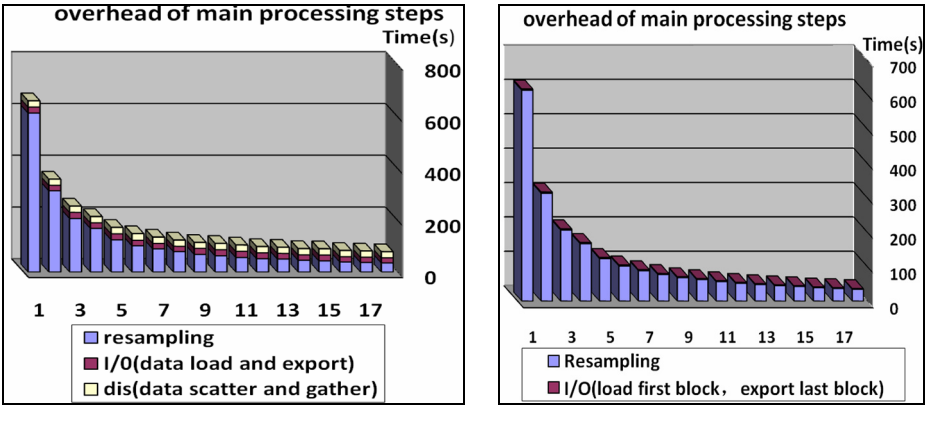



**Fig. 7.** Performance comparison




(a)Traditional parallel resampling algoithm (b) PIRA-PIO algoithm

**Fig. 8.** Time overhead of main processing steps

## 6 Conclusions and Future Work

The remote sensing image resampling is frequently used in various remote sensing image processing applications. It is compute-intensive, time-consuming and also poor in code reuse. In this article, to meet the performance challenge result from the I/O performance bottleneck, we propose PIRA-PIO algorithm. By adopting the data distribution and parallel I/O supported by parallel file system, using I/O hidden policy aims at eliminating I/O waiting time overhead, the performance of PIRA-PIO algorithm is significantly optimized. Through the evaluation and comparative analysis of the performance of PIRA-PIO algorithm and traditional parallel image resampling algorithm, we can see that the PIRA-PIO algorithm has gained an excellent parallel efficiency and linear speedup ratio. In addition, with the code reuse design, the flexibility of PIRA-PIO algorithm in various applications is greatly improved.

## References

1. Wittenbrink, C.M., Somani, A.K.: 2D and 3D Optimal Parallel Image Warping. J. Parallel Distrib. Comput. 25(2), 197–208 (1995)
2. Contassot-Vivier, S., Miguet, S.: A load-balanced algorithm for parallel digital image warping. International journal of pattern recognition and artificial intelligence 13(4), 445–463 (1999)
3. Jiang, Y.-h., Chang, Z.-m., Yang, X.: A Load-Balanced Parallel Algo-rithm for 2D Image Warping. In: Cao, J., Yang, L.T., Guo, M., Lau, F. (eds.) ISPA 2004. LNCS, vol. 3358, pp. 735–745. Springer, Heidelberg (2004)
4. Li, G., Ma, Y., Wang, J., Liu, D.: Preliminary Through-Out Research on Parallel-Based Remote Sensing Image Processing. In: Alexandrov, V.N., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2006. LNCS, vol. 3991, pp. 880–883. Springer, Heidelberg (2006)
5. Cerin, C., Jin, H.: Parallel I/O for cluster computing, London, Sterling, VA (2003)
6. Ruprecht, D., Muller, H.: Image Warping with Scattered Data. IEEE Computer Graphics and Applications 15(2), 37–43 (1995)
7. Warpenburg, M.R., Siegel, L.J.: SIMD image resampling. IEEE Transactions on Computers 31(10), 934–942 (1982)
8. Contassot-Vivier, S., Miguet, S.: A load-balanced algorithm for parallel digital image warping. International journal of pattern recognition and artificial intelligence 13(4), 445–463 (1999)
9. Lingjun, Z., Dingshen, L., Guoqing, L., wenyi, Z.: A Study Of High-Performance Precision Correction For Satellite Image. Remote Sensing For Land & Resources (1), 49–52 (2007)
10. Zhaoxia, F., Yan, H., Bo, Z.: An Automatic and Robust Image Mosaic Algorithm. Telecommunication Engineering 47(3), 55–58 (2007)
11. Webb, R.D.: Object Description Language [P]. United States Patent Application: 20030070159 (2003)