sustainability

Article

A Parallel and Optimization Approach for Land-Surface Temperature Retrieval on a Windows-Based PC Cluster

Bo Tie [1], Fang Huang [1,2,*], Jian Tao [3], Jun Lu [1] and Dongwei Qiu [4,*]

1 School of Resources & Environment, University of Electronic Science and Technology of China, 2006 Xiyuan Ave., West Hi-Tech Zone, Chengdu 611731, China; 2013180101017@std.uestc.edu.cn (B.T.); lujun9513@163.com (J.L.)

2 Institute of Remote Sensing Big Data, Big Data Research Center, University of Electronic Science and Technology of China, 2006 Xiyuan Road, West Hi-Tech Zone, Chengdu 611731, China

3 Texas A&M Engineering Experiment Station and High Performance Research Computing, Texas A&M University, College Station, TX 77843, USA; jtao@tamu.edu

4 School of Geomatics and Urban Spatial Informatics, Beijing University of Civil Engineering and Architecture, No. 1 Zhanlanguan Road, Xicheng District, Beijing100044, China

* Correspondence: hfhbhzp@uestc.edu.cn (F.H.); qiudw@bucea.edu.cn (D.Q.); Tel.: +86-158-8441-7588 (F.H.); +86-135-2070-2063 (D.Q.)

Received: 14 January 2018; Accepted: 22 February 2018; Published: 28 February 2018

Abstract: Land-surface temperature (LST) is a very important parameter in the geosciences. Conventional LST retrieval is based on large-scale remote-sensing (RS) images where split-window algorithms are usually employed via a traditional stand-alone method. When using the environment to visualize images (ENVI) software to carry out LST retrieval of large time-series datasets of infrared RS images, the processing time taken for traditional stand-alone servers becomes untenable. To address this shortcoming, cluster-based parallel computing is an ideal solution. However, traditional parallel computing is mostly based on the Linux environment, while the LST algorithm developed within the ENVI interactive data language (IDL) can only be run in the Windows environment in our project. To address this problem, we combine the characteristics of LST algorithms with parallel computing, and propose the design and implementation of a parallel LST retrieval algorithm using the message-passing interface (MPI) parallel-programming model on a Windows-based PC cluster platform. Furthermore, we present our solutions to the problems associated with performance bottlenecks and fault tolerance during the deployment stage. Our results show that, by improving the parallel environment of the storage system and network, one can effectively solve the stability issues of the parallel environment for large-scale RS data processing.

Keywords: land-surface temperature retrieving; ENVI interactive data language (IDL); parallel computing; windows-based PC cluster; message passing interface (MPI); Samba

1. Introduction

Land-surface temperature (LST) is a very important parameter in the geosciences, which plays a fundamental role in land–atmosphere interaction and is a key parameter in global environmental change in terms of its effect on global hydrology, ecology and biogeochemical processes [1–3]. LSTs also play an important role in the field of surface radiation and energy balance, drought monitoring, global and regional climate-change analysis, ecosystem modelling, surface-water thermal simulation, etc. [4–7]. In addition, remotely sensed LST is widely used in studying the surface urban heat island (SUHI), which can assess the SUHI effect [8] and find the correlation between LST and SUHI, e.g., the exact impact of temporal aggregation on LST and SUHI [9]. Liu et al. demonstrated that both biophysical and building-wall characteristics significantly influence the spatiotemporal variations of LST [10]. As a result, the question of how to obtain LST data/information economically and efficiently is of great interest to many disciplines in the natural sciences.

LST retrieval via remote-sensing (RS) methods can greatly improve the range of measurement and reduce the amount of work otherwise involved. Indeed, LST retrieval is a popular focus in current RS research. Spaceborne RS instruments that obtain high-precision LST data have been used in the field of quantitative RS research since the 1980s [11]. At present, the RS data that are applied to LST retrieval are provided by several instruments including the Thematic Mapper/Enhanced Thematic Mapper (TM/ETM+), the Advanced Spaceborne Thermal Emission and reflection Radiometer (ASTER), the Moderate-ResOlution Imaging Spectroradiometer (MODIS), and the Advanced Very High Resolution Radiometer (AVHRR).

At the same time, the split-window algorithm (SWA) for retrieving LSTs from satellite thermal infrared RS data, the single-window algorithm/single-channel algorithm, and the temperature/emissivity separation algorithm have also been established [12–15]. However, most split-window algorithms are aimed at processing National Oceanic and Atmospheric Administration (NOAA) and AVHRR data, and a long time series of LST products is lacking. In addition, the coefficients of the split-window algorithm are mostly "local", i.e., the algorithm coefficients depend on specific research areas and sensors. Therefore, our team members performed a large-scale radiation transmission simulation in which they fitted and tested more than 10 kinds of commonly used split-window algorithms. In their analysis, they selected nine split-window algorithms with high precision, low sensitivity and practicability, and built an integrated algorithm that used Bayesian weighted-model averaging (BMA) [16].

Incorporating global LST retrieval with an integrated algorithm based on the BMA model often involves a large number of long time-scale RS data calculations. To perform these calculations, the interactive data language (IDL) code is used, and several components have been specifically developed in the IDL environment to visualize images (ENVI) by the Exelis Visual Information Solutions (VIS) corporation. The traditional stand-alone IDL program is not capable of such tasks [17,18], and as the amount of RS data increases, it is difficult to process the data quickly with IDL programs in stand-alone environments. Thus, Exelis VIS offers the ENVI services engine (ESE) to provide IDL and ENVI image-processing services on a cluster or cloud-computing environment [19]. However, for real-world applications, an RS image-processing program usually contains one or more algorithms. If a parallel algorithm is implemented within the IDL program, it will take a lot of development time. Using ESE does save on development time, but commercial products are expensive and can be a financial burden. Fortunately, the emergence and application of different high-performance computing (HPC) technologies, e.g., the cluster-based parallel computing [20–24], graphics processing unit (GPU) based computing [25–29], and cloud computing [30–36] etc., makes large-scale data processing possible, which can greatly improve processing efficiency.

HPC systems are usually based on Linux. Existing LST-retrieval software needs to use IDL and other related, dependent environments. In upgraded versions of the IDL language environment, the latest version of ENVI is either unavailable or troublesome to deploy on Linux platforms. On the other hand, ENVI works well in the Windows operating system, and the widespread use of Windows makes it easier to build clusters of existing decentralized resources, which can save on costs and improve resource utilization.

There are a few existing studies that have used Windows-based clusters to develop their parallel algorithm or for parallel data processing. For example, Pan et al. used spare computers to form a "private cloud" in order to achieve a multi-computer parallel-computing framework for application in geophysical exploration [37]. Zheng used a Windows-based Beowulf cluster and a message passing interface (MPI) to study the application of parallel computing to seismic-damage analysis with RS images [38]. Moreover, Tie et al. applied parallel-computing technology and an MPI in a Windows cluster to batch process RS images, and proposed a simple parallel scheme for LST retrieval [39].

Unfortunately, this investigation only proposed the basics of the parallel-processing method and there were still some unaddressed issues such as storage and performance bottlenecks and fault tolerances.

In summary, to meet the demands of LST retrieval from thermal infrared data, and improve the efficiency of the retrieval, our aim is to use MODIS time-series RS data for LST retrieval. We propose a parallel algorithm for LST retrieval at the process level using an MPI parallel-programming model, which enables us to perform high-performance LST retrieval in a distributed memory environment with the Windows operating system. Furthermore, our research aims to solve the aforementioned, unaddressed problems present in previous study.

In addition to the LST parameter in global environmental and climate change, there are some other surface-temperature (ST) parameters such as sea-surface temperature (SST), lake-surface temperature, etc. The large-scale retrieval of these ST parameters are also applied to the large time-series RS data considered here. Compared with the retrieval of LSTs, existing research on SST retrieval has been around for a few decades. In 1980, McMillin pioneered the split-window approach to calculate SSTs on the basis of the 4th and 5th channels of AVHRR [40]. In 1985, McClain et al. proposed a single, linear multi-channel sea surface temperature method [41]. In 1999, Kumar et al. proposed the Miami pathfinder algorithm for SST that is suitable for processing MODIS data [42]. In 2004, Qin proposed a single-window method, specifically for the infrared channel of the TM6 satellite, which can also be applied to SST retrieval [43]. In recent years, many scholars have also studied the various factors that influence the SST retrieval procedure [44,45].

Meanwhile, lake-surface temperatures are monitored by satellites and numerous related works have been published in recent years. For instance, Livingstone used an RS image-retrieval method to reveal the relationship between the temperature of a lake in Australia and the local climate over a period of 80 years [46]. Pour et al. used MODIS images to retrieve the temperature of frozen Arctic lakes [47]. Moreover, Woolway et al. showed the relevance of lake-surface temperatures in relation to LST retrieval as it can be used to compare with LSTs, which is common in climatology [48]. Actually, all these algorithms have been improved and applied to lake-surface temperature retrieval based on LST or SST retrieval algorithms [49]. However, the parameters of different types of minerals vary from region to region, and from lake to lake. Thus, the required parameters for modelling one type of environment are often incompatible for modelling similar environments in different areas. Therefore, how to select parameters that are more suitable, and hence make the lake-surface temperature retrieval more accurate, has been a difficult obstacle to overcome in these studies. As such, our designed approach that adopts HPC for global LST retrieval by using large-scale RS data processing will, hopefully, be beneficial to the field, and can be applied to these, and others, ST-retrieval applications.

2. Background and Experimental Data Issue

This work was supported by a National High-Technology Research and Development Program (863) entitled "Generation and application of global products of essential land variables of global ecological system and surface-energy balance". The entire project aims to use multi-source RS data to produce global products that represent essential land variables, and which in turn can provide a database and technical support for researchers to make decisions about global change. Our results include: (1) the global products of essential land variables for 33 years (from 1982 to 2014), including the leaf-area index, emissivity, surface albedo, and photosynthetically active radiation; and (2) another eight products for four representative years (1983, 1993, 2003 and 2013), which involve shortwave radiation downstream of photosynthetically active radiation, LSTs, net long-wave radiation, net radiation (daytime), vegetation coverage, gross primary productivity, and latent heat [24]. Within this big project, our work is mainly focused on creating a new and integrated LST-retrieval algorithm that is suitable for generating long time series of LST products based on RS data, because these products have extremely important practical value for climate-change modeling, surface radiation and regional/global energy balance. In our sub-project, we are required to generate day-to-day LST products for a total of four periods (the years 1983, 1993, 2003 and 2013). The spatial resolution of the data in the years 1983 and 1993 is 5 km, while the others (2003, 2013) are 1 km. To carry out the task, we took the following steps: (1) based on a large-scale radiation transmission simulation, we selected several LST-retrieval algorithms from the existing split-window algorithms. These selected algorithms have advantages such as high precision, low sensitivity on the initial values of the inputted parameters, and highly practicability; (2) then, we built an integrated algorithm with a BMA model, as the BMA method has several advantages related to the integration of surface long-wave radiation models [50] and evapotranspiration model integration [51].

We note that, in the data-processing stage, we need to process four years' worth of data, while there are ~10 scenes of data from each day. To verify the feasibility of our approach quickly, we needed to experiment with some part of the data in advance in order to obtain a complete and reliable flow for our proposed HPC method. In this study, we chose only one scene from a single day of data. Therefore, we used 64 days of data for the experiments. Because this paper focuses on verifying the feasibility of the method, and it is impractical to demonstrate using the entire project data in this paper, the experiment data investigated here is only a small part of the entire data collected by the whole project.

3. Parallel Implementation of Land-Surface Temperature (LST) Retrieval

3.1. Fundamentals of LST Retrieval

At present, for spaceborne thermal infrared RS data with high time resolution that utilize two or more thermal infrared bands, the most suitable choice of algorithm for LST retrieval is the split-window algorithm [14]. This algorithm can be used for many different types of data including NOAA AVHRR, Environmental Satellite Advanced Along-Track Scanning Radiometer (ENVISAT AATSR), Terra/Aqua MODIS and Suomi National Polar-orbiting Partnership Visible Infrared Imaging Radiometer Suite (NPP VIIRS), and so on. The SWA is mainly based on the difference of atmospheric effects in two adjacent thermal infrared bands. Since the 1970s, the academic community has proposed dozens of SWAs. These algorithms have great similarity in form, and their general form can be summarized as:

$$T_s = A_0 + A_1 T_{11} + A_2 T_{12} \tag{1}$$

where, T_s is the land surface temperature; T_{11}, T_{12} are the channel brightness temperature at channel 11 and 12 respectively; and A_i (i = 0, 1, 2) is the algorithm factor.

The key to successfully using an SWA is to determine the appropriate input parameters and their values. The input parameters are varied, and they have different levels of complexity. Some algorithms only require the channel brightness temperature, while some external parameters, such as air vapor content, surface emissivity, near-surface temperature and land-cover type, are needed by other algorithms.

The strategy of building the LST-retrieval algorithm with different data sets is as follows: (1) select a variety of widely used SWAs to construct an algorithm library, where the chosen algorithms have their own forms and different input parameters; (2) build a representative atmospheric profile database, carry out a radiation transmission simulation and build training data sets, and then check the sensitivity of the data sets and analyze them; (3) determine the algorithm coefficients by using the simulated data set, determine the optimal algorithm (or combination of algorithms), and achieve the sensitivity analysis of the algorithm; and (4) based on real RS data, determine the calculation method of the input parameters of the algorithm.

In this paper, the global LST retrieval algorithm, which is based on an SWA, was applied to MODIS data as an example, where the entire technical roadmap of LST retrieval is shown in Figure 1. In addition, Figure 2 provides the detailed processing steps of the SWA.

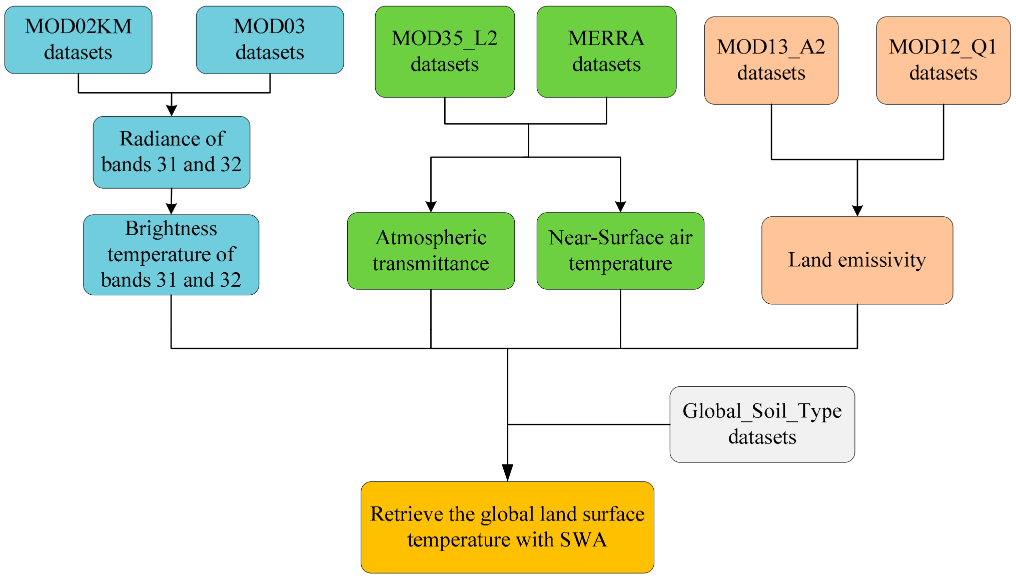
MOD02KM datasets
MOD03 datasets
MOD35_L2 datasets
MERRA datasets
MOD13_A2 datasets
MOD12_Q1 datasets
Radiance of bands 31 and 32
Brightness temperature of bands 31 and 32
Atmospheric transmittance
Near-Surface air temperature
Land emissivity
Global_Soil_Type datasets
Retrieve the global land surface temperature with SWA

Figure 1. Entire technical roadmap of global land-surface temperature (LST) retrieval.

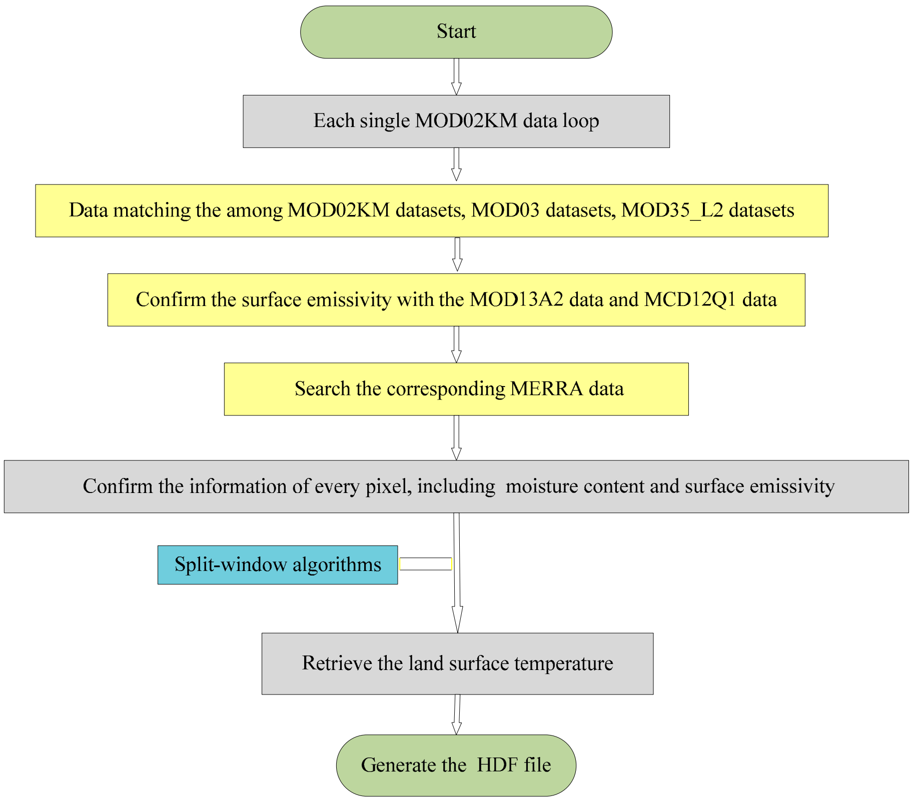
Start
Each single MOD02KM data loop
Data matching the among MOD02KM datasets, MOD03 datasets, MOD35_L2 datasets
Confirm the surface emissivity with the MOD13A2 data and MCD12Q1 data
Search the corresponding MERRA data
Confirm the information of every pixel, including moisture content and surface emissivity
Split-window algorithms
Retrieve the land surface temperature
Generate the HDF file

Figure 2. Flowchart of the LST retrieval algorithm.

From Figure 2, the processing procedure can be divided into three steps: (1) data indexing; (2) data matching; and (3) data inversion. The entire program starts with the MOD02KM data, and then assigns the data to be processed in turn. To match the other data sets based on the MOD02KM data, it will obtain some information from the data-matching process, such as the radiance brightness, longitude and latitude and so on. After the SWA finishes running, we put the retrieved LSTs into hierarchical data file (HDF) format.

3.2. Parallel Design and Implementation of the Global LST-Retrieval Algorithm

As IDL does not support the distributed computing technique, in order to implement cluster-based parallel computing we used other languages to enable inter-process communication. One approach is to use IDL programs to call encapsulated MPI communication functions. Another is to call IDL programs directly from other languages. The latter is currently common practice in the field. In our implementation, we used the C language to develop MPI programs for underlying communication, while invoking IDL programs to process the data. The parallel program uses the MPICH2 library [52], which is a popular implementation of the MPI standard, to support multi-node parallel processing. By separating the MPI-based communication programs and the IDL-based data-processing programs, we make the whole development process more suitable for a multidisciplinary collaboration and easier to manage.

3.2.1. Parallelization of the Serial Algorithm

As the serial algorithm is designed for stand-alone environments, the design and operation of the algorithm has been optimized to the working environment of a single computer. It is necessary to refactor the serial program and adjust it to the distributed computing environment. This transformation includes path modification of the original IDL program and its input data, as well as the modularization of the IDL program itself. These steps are described as follows.

(1) Path Modification

For LST retrieval, the absolute paths of the input and output data need to be reconstructed first. By taking advantage of the application-programming interface of the MPI, we decided to modify the input and output path of the data and make the parameters suitable for the application-programming interface.

(2) Serial Algorithm Modularization

To run LST retrieval in parallel on a Windows-based cluster, we eliminated the traditional IDL graphical interaction required at run time and the IDL console interface to improve the parallelization efficiency. We packaged the IDL programs in the "sav" file format and utilized the "cmd" command to call the IDL sub-routine "idlrt.exe" to run the programs in the "sav" file. Thus, the packaged programs basically run without the IDL runtime environment. By calling only one sub-routine with "cmd" together with the settings of the relevant environment and parameters, this approach greatly reduces the burden involved in running IDL programs. This lays the ground for the parallel optimization of the serial algorithm.

3.2.2. Design and Implementation of the Parallel Algorithm

The primary parallelization strategy for IDL programs is data parallelism. One can divide the RS data to be processed into units of days. By organizing the configuration (input) file of the daily data, one can divide the data using the master process and assign them to individual worker processes. In general, the master process obtains the overall information about the data to be processed, and then assigns the input file for the sliced daily data to each worker process. Each worker process then obtains the respective data file, reads data that need to be processed, processes the data, and then obtains the result. The process is shown in Figure 3.

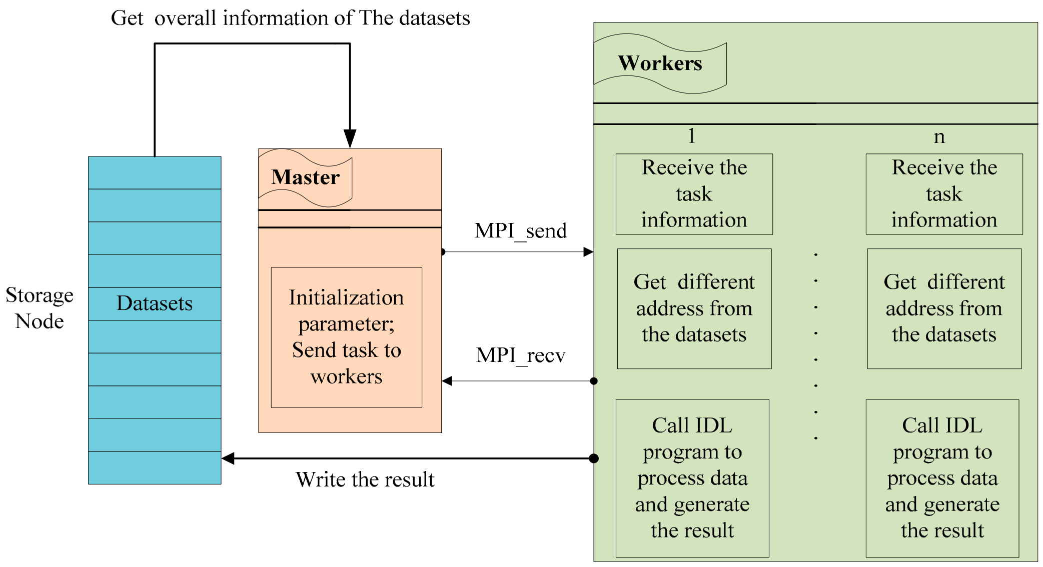
Get overall information of The datasets
Storage Node
Datasets
Master
Initialization parameter; Send task to workers
MPI_send
MPI_recv
Write the result
Workers
1
n
Receive the task information
Get different address from the datasets
Call IDL program to process data and generate the result
Receive the task information
Get different address from the datasets
Call IDL program to process data and generate the result

Figure 3. Schematic of the parallel algorithm.

The detailed parallelization procedure is elaborated as follows:

(1) Front node (Master for short) initializes the running environment, obtains the path of the data, and resolves the specific tasks that will be distributed to the computing nodes (also called workers).
(2) Master broadcasts (assigns) data paths to the computing nodes, and the computing nodes find their respective task data in the storage nodes according to their respective paths.
(3) Computing nodes process the data. Each computing node will do their own subtask simultaneously. After the data processing completes, the computing nodes write the results to the storage nodes, and send their own status to the Master.
(4) Master analyzes the messages that it receives from the computing nodes and decides whether to continue assigning tasks to the computing nodes based on the status of the computing nodes and the number of remaining tasks.
(5) Repeat steps (2), (3), and (4) until all tasks are accomplished.
(6) Master ends the running environment and the program.

The parallel algorithm contains the following steps: (1) initialize the MPI environment; (2) obtain the rank (identification number) of each process; (3) the MPI program sends a message; (4) the MPI program receives a message; and (5) the MPI program terminates. Developers can add their own functionality to each of these steps. A flowchart of the implementation is shown in Figure 4.

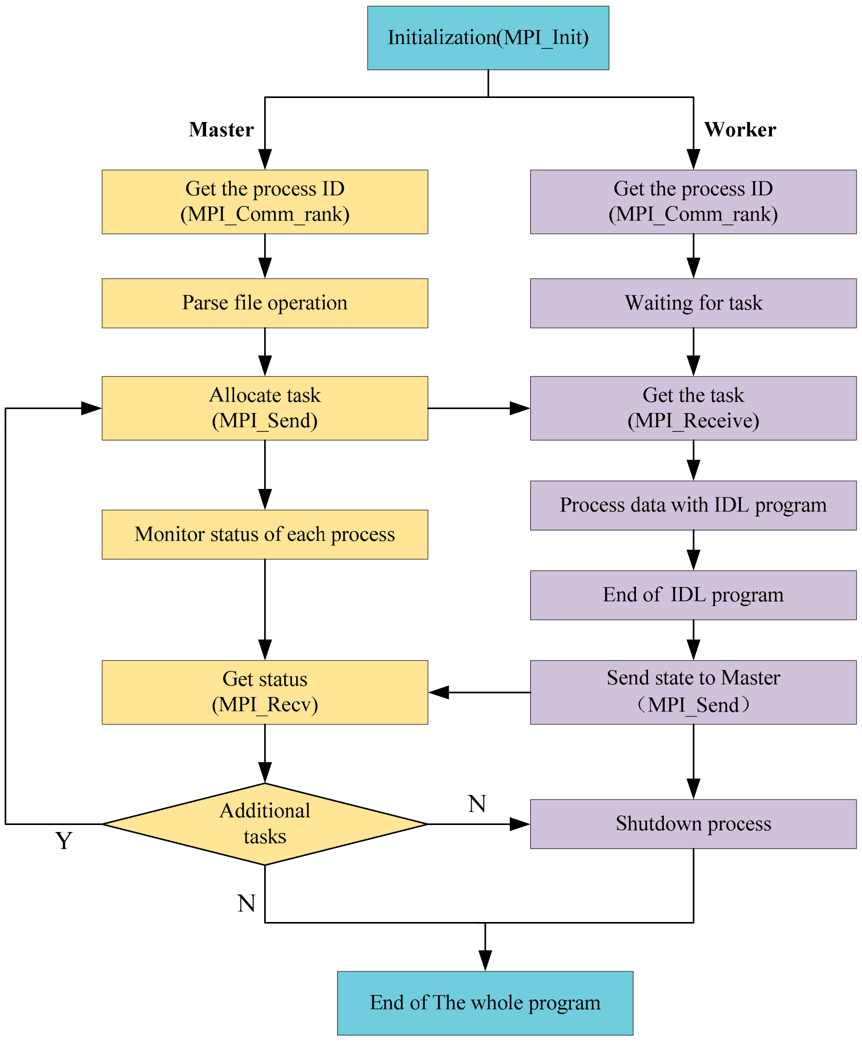

Figure 4. Workflow of the parallel implementation.

In the implementation, we assign one Master process and one or more worker processes depending on the runtime option from the user. Only one worker process is shown in the diagram.

4. Performance Testing and Evaluation

4.1. System Environment Configuration

We used PCs hosted in the lab of our school and established them into a Windows PC cluster. We shared a disk partition at the front node for storage and to provide data access through the shared folder. The configuration of each node is shown in Table 1.

Table 1. Cluster configuration.

Nodes	Operating System (OS)	Memory	Network	Processor	Number
Master	Windows 7	4 GB	100 M	Intel i3-3240	1
Computing nodes	Windows 7	4 GB	100 M	Intel i3-3240	4

The software environment is primarily configured for running MPI and ENVI IDL programs. The software configuration of the specific nodes is shown in Table 2.

Table 2. Software configuration on different nodes.

Software	MPICH2	ENVI	IDL
Version number	1.4.1	4.8	8.0
Installation location	Master & computing nodes	Master	Master

4.2. Performance Evaluation Metrics

To evaluate the performance of a parallel algorithm, the speed-up metric is usually used [53]:

$$Speedup = T_s / T_p \tag{2}$$

where, T_s is the run time of the serial program and T_p is the run time of the parallel program. The speed-up metric indicates how fast the parallel program is compared to the serial one. The greater the speed-up, the better the parallel program.

4.3. Experimental Results and Analysis

In our experiments, we selected 64 days of MODIS data from 2013. Based on the number of cores and the memory limitation per node, we were limited to four processes per computing node, and we recorded the total time for different number of machines and processes. Table 3 shows the time consumed by single and multiple nodes, where each node contains two and four MPI processes, respectively. When we further increased the number of computing nodes, or the number of MPI processes contained in one node, the processing procedure became unstable. It is possible that errors occur when too many MPI processes simultaneously read the same file from the storage node across the network as the number of processes increases.

Table 3. Time consumption in seconds.

P \ *n*	2	4
1	11,386	9765
2	5891	4785
3	3891	3317
4	3099	3002
5	2479	null
6	2140	null

n is the number of nodes, and *P* is the number of message-passing interface (MPI) processes running on each node; "null" means the experiment was not completed due to errors.

As shown in Table 4, we calculated the achieved speed-up metric for each case.

Table 4. Achieved speed-up.

P \ *n*	2	4	Optimal Speed-Up
1	1.24	1.47	1.47
2	2.40	2.95	2.95
3	3.63	4.26	4.26
4	4.70	4.71	4.71
5	5.69	null	5.69
6	6.60	null	6.60

n is the number of nodes, and *P* is the number of MPI processes running on each node.

From Tables 3 and 4, we can see that when the number of MPI processes on a single node is the same, the processing time gradually decreases as the number of nodes increases, and the acceleration ratio increases as the number of computing nodes increases. However, we can see that in the four-nodes—four-processes case, the time elapsed is reduced. By comparing the speed-up of the four-node case with the five-node and six-node cases, we can observe that when the total number of processes exceeds eight, simply increasing the number of processes per node will have less effect on the speed-up. Considering the hardware used in the tests, we expect that the network bandwidth was saturated for cases with more than 12 MPI processes. The results are also demonstrated in Figure 5.

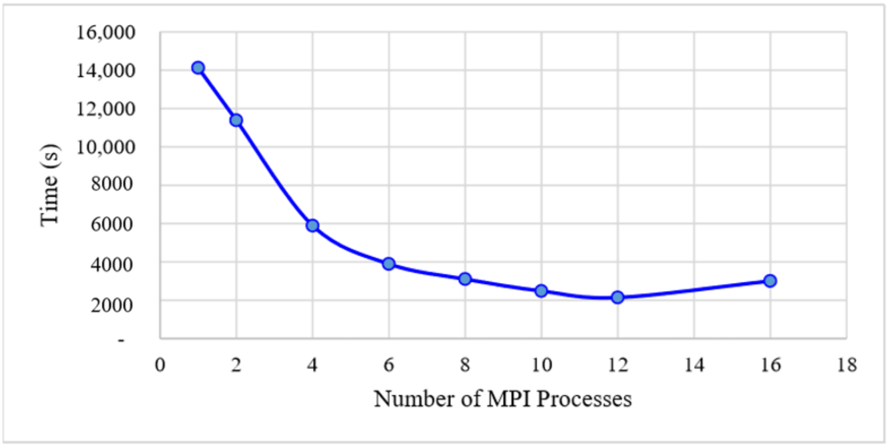

Figure 5. Time consumption versus the number of MPI processes.

An increase in the number of MPI processes will cause an increase in the number of reads and writes to the storage node, which results in a decrease in the computing performance of the nodes when the total bandwidth available cannot meet the algorithm's requirements.

From the overall experimental results, this method has been demonstrated to work in the full Windows 7 operating system environment. When one computer is used as a data-storage node and six more as computing nodes (two MPI processes per node) for distributed computing, compared to the traditional serial program running on a single machine, the overall performance is improved by almost a factor of seven.

5. Optimization Approaches of the Parallel LST Retrieval Algorithm

Based on the foregoing basic experiments, we achieved very good performance with a PC cluster running the Microsoft Windows 7 operating system. However, we noticed two problems:

(1) When we increased the number of computers for parallel LST retrieval, we found that the entire system's operating environment became less stable, and the computers were unable to find the assigned task files. As Windows 7 is used in all the experiments, data transfer between a storage node and the computing nodes used Windows 7 network file sharing for the local area network. Such data sharing can only support four or five computing nodes (four MPI processes per node) for parallel computing. Once the number of computing nodes goes beyond this, the entire system environment will either stall or collapse. This constraint fundamentally limits the number of computers involved in the deployment of parallel computing applications on PC clusters.

(2) According to the performance analysis of the basic experiment, we notice that more MPI processes (e.g., four nodes with four processes) has a lower speed-up metric than fewer MPI processes (e.g., five nodes with two processes). Meanwhile, when the number of MPI processes reaches 12, increasing the number of MPI processes does not have much effect on the speed-up. As explained at the end of previous section, we expect that the network bandwidth was saturated in those cases, and the total bandwidth available could not meet the requirements of our application.

Based on the foregoing analysis, in order to resolve the problems we modified the computing environment, especially the storage node, and improved the network connectivity.

5.1. Improvement Approach One: Modifying the Software Configuration of the Storage Node

As the Windows 7 operating system restricts the maximum number of connections to a storage node, we decided to use a different operating system. One option was a Windows server operating system, which is often used for enterprise-class management systems and can serve many users at the same time for data requests, sends, retrievals, and hard-disk access [54]. On the other hand, the Linux operating system is the de facto operating system for high-performance computing. Since Linux cannot share data directly with Windows, we used the Samba file services in Linux to enable data exchange with Windows [55].

We tested the Windows server system and the Samba server in Linux, respectively, on the storage node using 64 days of data. In the experiments, we set the maximum number of MPI processes per compute node to four, and tried 32-process and 64-process runs, and hence recorded the total time consumed. Figure 6 shows a diagram of the connection among the nodes. The number of computing nodes were either eight or 16 in our experiments.

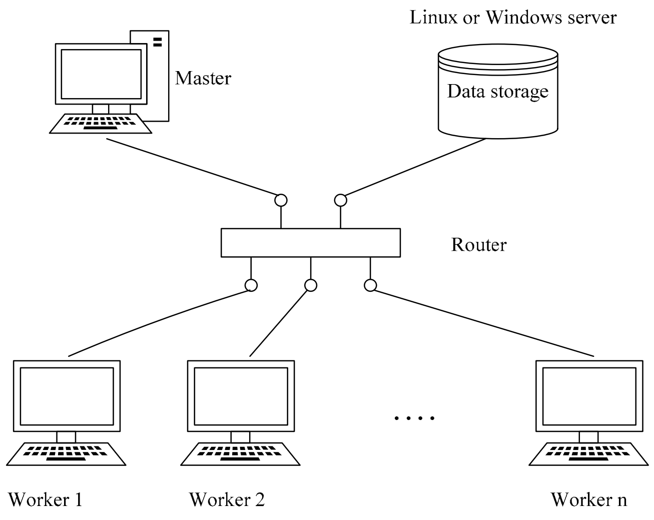

Figure 6. Diagram of the optimized parallel environment.

The node configurations of the two cases are shown in Tables 5 and 6, respectively.

Table 5. Case 1 nodes configuration.

Node Type	System	Memory	Network	Number
Master	Windows 7	4 G	1000 M	1
Computing node	Windows 7	4 G	1000 M	8 or 16
Storage node	Windows Server 2012	4 G	1000 M	1

Table 6. Case 2 nodes configuration.

Node Type	System	Memory	Network	Number
Master	Windows 7	4 G	1000 Mbps	1
Computing node	Windows 7	4 G	1000 Mbps	N
Storage node	Linux with Samba	4 G	1000 Mbps	1

In Case 1 we tested the performance of the Windows server installed on the storage node, while in Case 2 we tested the Linux Samba server. The software environment was primarily configured for running MPI and IDL programs. The configurations shown in Table 6 were used, and our experimental results are displayed in Table 7.

Table 7. Experimental results for Case 1 and Case 2. Two identical tests (Test #1 and Test #2) were carried out to ensure that no extreme cases appeared in the tests.

Experimental Conditions	Linux (Case 2) 32 Processes	Linux (Case 2) 64 Processes	WinServer (Case 1) 32 Processes	WinServer (Case 1) 64 Processes
Test #1 (s)	1782	1239	1483	1639
Test #2 (s)	1753	1342	1624	1612

From Table 7, it can be seen that the system-crash issues for excessive file-server visits have been solved. All the experiments finished successfully. By comparing the results for the two storage configurations, we observed that the Linux Samba server outperformed the Windows server when the number of MPI processes is large. However, the Windows server was slightly better when fewer processes are used.

5.2. Improvement Approach Two: Effects of Network Transmission Rates

In our parallelization method, an increase in the number of MPI processes results in increased synchronization of data acquisition and transmission. We are interested in verifying that the amount of data that needs to be transferred per unit of time increases with the increase in the number of MPI processes. Furthermore, we are interested in knowing if the network bandwidth of our previous configuration did not meet the computational requirements, thus limiting the increase in the speed-up. In this experiment, we used 100 Mbps and 1 Gbps network configurations to carry out the comparison.

The configuration of the experimental setup is the same as given in Section 4.1. No other configurations have been changed in this experiment, except that a different network configuration is used. The system connection diagram is shown in Figure 7.

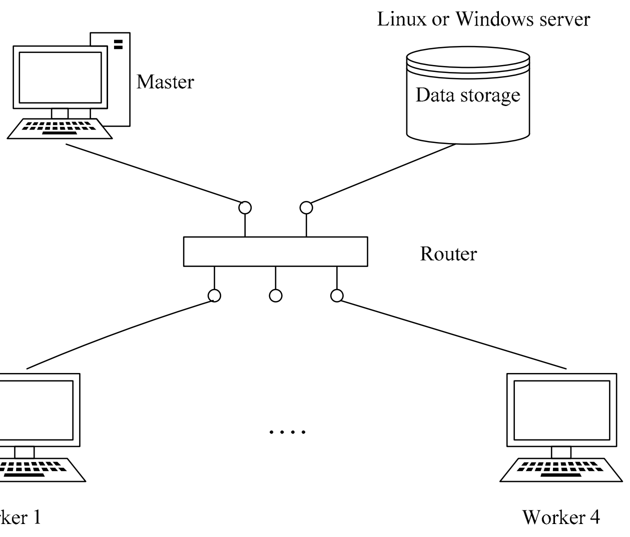

Figure 7. System diagram for testing with different network configurations.

We used the same PC cluster with different networking devices (see Tables 8 and 9) to process the data from 32 days, and we used four MPI processes per node.

Table 8. Case 3 node configuration.

Node Type	System	Memory	Internet	Number
Master	Windows 7	4 GB	100 Mbps	1
Computing node	Windows 7	4 GB	100 Mbps	4
Storage node	Windows 7	4 GB	100 Mbps	1

Table 9. Case 4 node configuration.

Node Type	System	Memory	Internet	Number
Master	Windows 7	4 GB	1 Gbps	1
Computing node	Windows 7	4 GB	1 Gbps	4
Storage node	Windows 7	4 GB	1 Gbps	1

As shown above, in Case 3 we used the 100 Mbps network, while in Case 4 we used the 1 Gbps network. Compliant with the aforementioned experiments, we obtained the experimental results of these cases (Tables 10 and 11; and Figures 8 and 9, respectively).

Table 10. Case 3 and Case 4 experimental results in seconds.

Total Number of MPI Processes: 4 (Processes/Node) × (Number of Nodes)	4 × 1	4 × 2	4 × 3	4 × 4	Serial
100 M Network (Case 3)	4684	3099	2762	3002	14,127
1 G Network (Case 4)	3714	2631	2728	3215	

Table 11. Speed-up for the different network configurations.

Total Number of MPI Processes: 4 (Processes/Node) × (Number of Nodes)	4 × 1	4 × 2	4 × 3	4 × 4	Serial
100 M Network	3.02	4.56	5.11	4.71	1
1 G Network	3.80	5.40	5.18	4.39	1

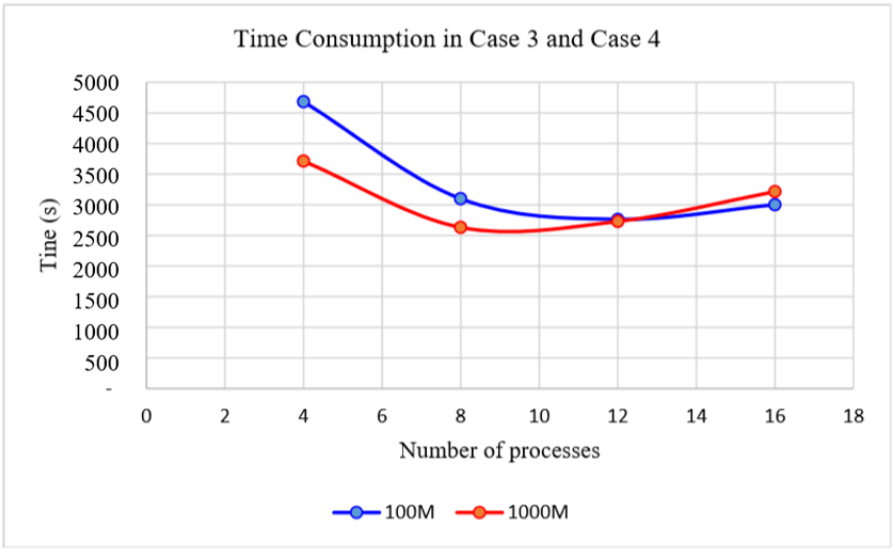

Figure 8. Time consumption in Case 3 and Case 4 for different network configurations.

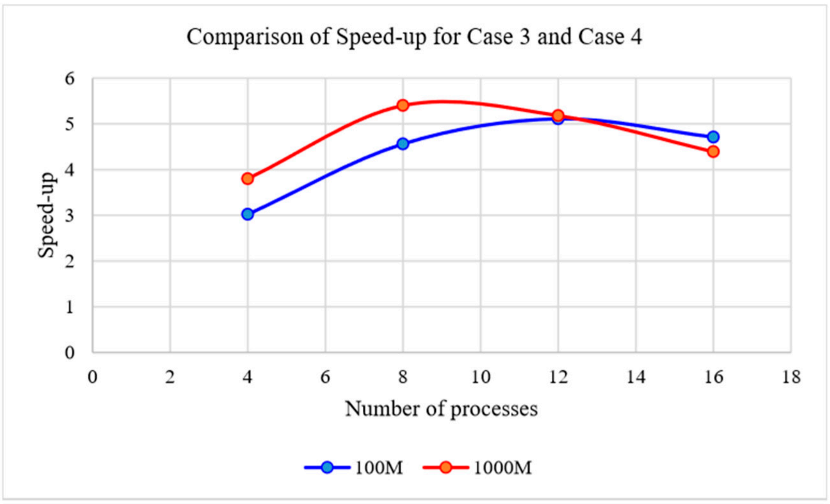

Figure 9. Speed-up of Case 3 and Case 4 for different network configurations.

The performance of the parallel implementation is greatly improved when the quality of the network configuration is upgraded (Figure 8). Combined with Figure 9, we find that a faster network transmission speed results in a better speed-up. This also demonstrates the reason why the obtained speed-up of four nodes with four MPI processes is much lower than that of five nodes with two MPI processes (i.e., Table 4). The network transmission speed affects the program's ability to process real-time data; especially when a large amount of data needs to be dealt with, the impact of network speed cannot be ignored. However, the achieved speed-up will change only slightly with increased bandwidth when the parallel algorithm reaches its optimal theoretical performance state.

In summary, the experimental results show that either the Windows server or the Linux Samba server can be a good solution to resolve the limitation in the number of computing nodes in a Windows-based PC cluster with a shared file server. Additionally, an optimized and upgraded network configuration can also speed up the system.

6. Conclusions and Future Directions

Considering that most of the coefficients of SWAs have "local" characteristics, this paper uses a BMA model to build an integrated algorithm on top of the traditional serial-retrieving program that forms the basis of an MPI-based parallel LST-retrieval algorithm. Our experimental results show that the parallel algorithm can effectively shorten LST retrieval time. By selecting a reasonable number of processes and using four computers, the maximum acceleration ratio is close to five. To optimize the parallel algorithm, we improved the network transmission and the data storage in the implementation of the algorithm by: (1) analyzing the bottleneck (i.e., network configuration) in the workflow and upgrading the networking devices; (2) analyzing the system configuration of the data storage node by carrying out a comparison between a Windows server and a Linux Samba server to replace the Windows 7 operating system on the storage node. Our experimental results show that the time of parallel-task processing can be significantly reduced by upgrading the networking devices, and the acceleration ratio is significantly enhanced. Using either the Windows server or the Linux Samba server to replace the existing storage-node operating system solves the crashing problem caused by excessive connections from computing nodes. However, the results of the parallel performance still leave plenty of room for improvement. Based on this study, further improvements can be carried out in the following areas:

(1) In the experiments, we observed that the speed-up in the 1 Gbps network case is much lower than that of the 100 Mbps case when there are over 12 MPI processes. The lower speed-up could be caused by the dramatic improvement in performance of the serial program in the 1 Gbps case.

The greatly reduced running time of the serial program, which serves as the nominator to calculate speed-up, helps to reduce the speed-up calculated for the 1 Gbps case. Further investigations are necessary.

(2) In the comparison test of the replaced storage node in Section 5.1, it was found that the speed-up slows down when the number of computing nodes increases. We suspect that a single storage node cannot meet the needs of a large number of computing nodes for reading and writing data. We could consider using a distributed file system to replace the single-data storage node in order to further improve the overall performance of the system.

Acknowledgments: This study was supported mainly by the National Key Research and Development Program of China (No. 2017YFB0503700), the Fundamental Research Funds for the Central Universities (Grant No. ZYGX2015J111), Hubei Provincial Key Laboratory of Intelligent Geo-information Processing (China University of Geosciences) (Grant No. KLIGIP2016A03), the Key Laboratory of Spatial Data Mining and Information Sharing of the Ministry of Education, Fuzhou University (2017LSDMIS03), and also the National Science Foundation of the United States (Award No. 1723292).

Author Contributions: Fang Huang conceived of and designed the experiments and revised the paper. Bo Tie performed the experiments and wrote the paper. Jun Lu also performed the experiments. Jian Tao and Dongwei Qiu analyzed the data and made key modifications to the paper.

Conflicts of Interest: The authors declare that they have no conflicts of interest.

References

1. Running, S.W.; Justice, C.O.; Salomonson, V.; Hall, D.; Barker, J.; Kaufmann, Y.J.; Strahler, A.H.; Huete, A.R.; Muller, J.P.; Vanderbilt, V.; et al. Terrestrial remote-sensing science and algorithms planned for EOS MODIS. *Int. J. Remote Sens.* **1994**, *15*, 3587–3620. [CrossRef]
2. Vining, R.C.; Blad, B.L. Estimation of sensible heat-flux from remotely sensed canopy temperatures. *J. Geophys. Res. Atmos.* **1992**, *97*, 18951–18954. [CrossRef]
3. Caselles, V.; Sobrino, J. Determination of frosts in orange groves from NOAA-9 AVHRR data. *Remote Sens. Environ.* **1989**, *29*, 135–146. [CrossRef]
4. Yang, K.; He, J.; Tang, W.J.; Qin, J.; Cheng, C.C.K. On downward shortwave and longwave radiations over high altitude regions: Observation and modeling in the Tibetan Plateau. *Agric. For. Meteorol.* **2010**, *150*, 38–46. [CrossRef]
5. Zhou, J.; Chen, Y.; Wang, J.; Zhan, W. Maximum nighttime urban heat island (UHI) intensity simulation by integrating Remotely Sensed data and meteorological observations. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2011**, *4*, 138–146. [CrossRef]
6. Zhou, J.; Chen, Y.H.; Zhang, X.; Zhan, W.F. Modelling the diurnal variations of urban heat islands with multi-source satellite data. *Int. J. Remote Sens.* **2013**, *34*, 7568–7588. [CrossRef]
7. Zhou, J.; Zhang, X.D.; Zhan, W.F.; Gottsche, F.M.; Liu, S.M.; Olesen, F.S.; Hu, W.X.; Dai, F.N. A thermal sampling depth correction method for land surface temperature estimation from satellite passive microwave observation over barren land. *IEEE Trans. Geosci. Remote Sens.* **2017**, *55*, 4743–4756. [CrossRef]
8. Zhang, Y.; Bai, Z.; Liu, W. Assessing the Surface Urban Heat Island Effect in Xining, China. In *Geo-Informatics in Resource Management and Sustainable Ecosystem*; Bian, F., Xie, Y., Cui, X., Zeng, Y., Eds.; Springer: Berlin/Heidelberg, Germany, 2013; Volume 398, pp. 264–273.
9. Hu, L.; Brunsell, N.A. The impact of temporal aggregation of land surface temperature data for surface urban heat island (SUHI) monitoring. *Remote Sens. Environ.* **2013**, *134*, 162–174. [CrossRef]
10. Liu, W.; Feddema, J.; Hu, L.; Zung, A.; Brunsell, N. Seasonal and Diurnal Characteristics of Land Surface Temperature and Major Explanatory Factors in Harris County, Texas. *Sustainability* **2017**, *9*, 2324. [CrossRef]
11. Wan, Z.M.; Dozier, J. A generalized split-window algorithm for retrieving land-surface temperature from space. *IEEE Trans. Geosci. Remote Sens.* **1996**, *34*, 892–905.
12. Qin, Z.; Dall'Olmo, G.; Karnieli, A.; Berliner, P. Derivation of split window algorithm and its sensitivity analysis for retrieving land surface temperature from NOAA-advanced very high resolution radiometer data. *J. Geophys. Res. Atmos.* **2001**, *106*, 22655–22670. [CrossRef]

13. Gillespie, A.; Rokugawa, S.; Matsunaga, T.; Cothern, J.S.; Hook, S.; Kahle, A.B. A temperature and emissivity separation algorithm for Advanced Spaceborne Thermal Emission and Reflection Radiometer (ASTER) images. *IEEE Trans. Geosci. Remote Sens.* **1998**, *36*, 1113–1126. [CrossRef]
14. Li, Z.L.; Tang, B.H.; Wu, H.; Ren, H.Z.; Yan, G.J.; Wan, Z.M.; Trigo, I.F.; Sobrino, J.A. Satellite-derived land surface temperature: Current status and perspectives. *Remote Sens. Environ.* **2013**, *131*, 14–37. [CrossRef]
15. Tang, B.H.; Shao, K.; Li, Z.L.; Wu, H.; Nerry, F.; Zhou, G.Q. Estimation and validation of land surface temperatures from Chinese second-generation polar-orbit FY-3A VIRR Data. *Remote Sens.* **2015**, *7*, 3250–3273. [CrossRef]
16. Kass, R.E.; Raftery, A.E. Bayes Factors. *J. Am. Stat. Assoc.* **1995**, *90*, 773–795. [CrossRef]
17. Zhang, X.; Ding, F.; Peng, X.L.; Wu, W.F.; Fan, P.Y. Fast retrieval of land surface emissivity from Landsat data through IDL programming. In Proceedings of the 2014 3rd International Workshop on Earth Observation and Remote Sensing Applications (EORSA), Changsha, China, 11–14 June 2014.
18. Ou, W.H.; Su, W.; Wu, C.; Zhu, Z.Z.; Li, Y.M.; Shen, S. Drought monitoring based on the vegetation temperature condition index by IDL language processing method. In *Computer and Computing Technologies in Agriculture V, Part III*; Li, D.L., Chen, Y.Y., Eds.; Springer-Verlag: Berlin, Germany, 2012; Volume 370, pp. 43–49.
19. ENVI Services Engine. Available online: https://www.harrisgeospatial.com/docs/enviservicesengine.html (accessed on 24 February 2018).
20. Armstrong, M.P.; Marciano, R.J. Local interpolation using a distributed parallel supercomputer. *Int. J. Geogr. Inf. Syst.* **1996**, *10*, 713–729. [CrossRef]
21. Lanthier, M.; Nussbaum, D.; Sack, J.R. Parallel implementation of geometric shortest path algorithms. *Parallel Comput.* **2003**, *29*, 1445–1479. [CrossRef]
22. Achhab, N.B.; Raissouni, N.; Azyat, A.; Chahboun, A.; Lahraoua, M. High performance computing software package for multitemporal Remote-Sensing computations. *Int. J. Eng. Technol.* **2010**, *2*, 360–365.
23. Huang, F.; Liu, D.S.; Tan, X.C.; Wang, J.; Chen, Y.P.; He, B.B. Explorations of the implementation of a parallel IDW interpolation algorithm in a Linux cluster-based parallel GIS. *Comput. Geosci.* **2011**, *37*, 426–434. [CrossRef]
24. Huang, F.; Zhou, J.; Tao, J.; Tan, X.; Liang, S.; Cheng, J. PMODTRAN: A parallel implementation based on MODTRAN for massive remote sensing data processing. *Int. J. Digit. Earth* **2016**, *9*, 819–834. [CrossRef]
25. Chen, D.; Li, D.; Xiong, M.; Bao, H.; Li, X. GPGPU-Aided Ensemble Empirical-Mode Decomposition for EEG Analysis during Anesthesia. *IEEE Trans. Inf. Technol. Biomed.* **2010**, *14*, 1417–1427. [CrossRef] [PubMed]
26. Chen, D.; Wang, L.; Tian, M.; Tian, J.; Wang, S.; Bian, C.; Li, X. Massively parallel Modelling & Simulation of large crowd with GPGPU. *J. Supercomput.* **2013**, *63*, 675–690.
27. Bernabé, S.; Lopez, S.; Plaza, A.; Sarmiento, R. GPU Implementation of an Automatic Target Detection and Classification Algorithm for Hyperspectral Image Analysis. *IEEE Geosci. Remote Sens. Lett.* **2013**, *10*, 221–225. [CrossRef]
28. Liu, P.; Yuan, T.; Ma, Y.; Wang, L.; Liu, D.; Yue, S.; Kolodziej, J. Parallel processing of massive remote sensing images in a GPU architecture. *Comput. Inf.* **2014**, *33*, 197–217.
29. Huang, F.; Tao, J.; Xiang, Y.; Liu, P.; Dong, L.; Wang, L.Z. Parallel compressive sampling matching pursuit algorithm for compressed sensing signal reconstruction with OpenCL. *J. Syst. Architect.* **2017**, *72*, 51–60. [CrossRef]
30. Wang, L.; Tao, J.; Ranjan, R.; Marten, H.; Streit, A.; Chen, J.; Chen, D. G-Hadoop: MapReduce across distributed data centers for data-intensive computing. *Future Gener. Comput. Syst.* **2013**, *29*, 739–750. [CrossRef]
31. Wang, L.; Chen, D.; Liu, W.; Ma, Y.; Wu, Y.; Deng, Z. DDDAS-based parallel simulation of threat management for urban water distribution systems. *Comput. Sci. Eng.* **2014**, *16*, 8–17. [CrossRef]
32. Deng, Z.; Hu, Y.; Zhu, M.; Huang, X.; Du, B. A scalable and fast OPTICS for clustering trajectory big data. *Clust. Comput.* **2015**, *18*, 549–562. [CrossRef]
33. Hu, C.; Zhao, J.; Yan, X.; Zeng, D.; Guo, S. A MapReduce based Parallel Niche Genetic Algorithm for contaminant source identification in water distribution network. *Ad Hoc Netw.* **2015**, *35*, 116–126. [CrossRef]
34. Wang, Y.; Liu, Z.; Liao, H.; Li, C. Improving the performance of GIS polygon overlay computation with MapReduce for spatial big data processing. *Clust. Comput.* **2015**, *18*, 507–516. [CrossRef]

35. Yu, T.; Dou, M.; Zhu, M. A data parallel approach to modelling and simulation of large crowd. *Clust. Comput.* **2015**, *18*, 1307–1316. [CrossRef]
36. Huang, F.; Zhu, Q.; Zhou, J.; Tao, J.; Zhou, X.; Jin, D.; Tan, X.; Wang, L. Research on the parallelization of the DBSCAN clustering algorithm for spatial data mining based on the Spark platform. *Remote Sens.* **2017**, *9*, 1301. [CrossRef]
37. Pan, Y.; Ma, Q.; Du, Q. Multi-computer parallel-computing framework for application in geophysical exploration. *Prog. Geophys.* **2017**, *32*, 891–897.
38. Zheng, Y. *Study on Application of Parallel Computing to Seismic Damage Analysis from Remote Sensing Images*; Institute of Earthquake Science, China Earthquake Administration: Beijing, China, 2010.
39. Tie, B.; Huang, F.; Tao, J.; Zhou, J. Parallel land surface temperature retrieval with Windows cluster. *Int. J. Adv. Comput. Technol.* **2017**, *9*, 1–9.
40. McMillin, L.M. The split window retrieval algorithm for sea surface temperature derived from satellite measurements A2—Deepak, ADARSH. In *Remote Sensing of Atmospheres and Oceans*; Academic Press: New York, NY, USA, 1980; pp. 437–455.
41. McClain, E.P.; Pichel, W.G.; Walton, C.C. Comparative performance of AVHRR-based multichannel sea surface temperatures. *J. Geophys. Res. Oceans* **1985**, *90*, 11587–11601. [CrossRef]
42. Kumar, A.; Minnett, P.; Podesta, G.; Evans, R.; Kilpatrick, K. Analysis of Pathfinder SST algorithm for global and regional conditions. *J. Earth Syst. Sci.* **2000**, *109*, 395–405. [CrossRef]
43. Qin, Z.; Li, W.; Xu, B.; Zhang, W. Estimation method of land surface emissivity for retrieving land surface temperature from Landsat TM6 data. *Adv. Mar. Sci.* **2004**, *z1*, 129–137.
44. Kilpatrick, K.A.; Podesta, G.; Walsh, S.; Williams, E.; Halliwell, V.; Szczodrak, M.; Brown, O.B.; Minnett, P.J.; Evans, R. A decade of sea surface temperature from MODIS. *Remote Sens. Environ.* **2015**, *165*, 27–41. [CrossRef]
45. Xiong, P.; Zhu, L.; Gu, X.; Zhao, L.; Yu, T.; Meng, Q.; Li, J.; Zhang, F. Sea water temperature retrieval model for Daya Bay based on HJ-1B thermal infrared remote sensing data and its application. *Remote Sens. Land Resour.* **2014**, *26*, 132–138.
46. Livingstone, D.M.; Dokulil, M.T. Eighty years of spatially coherent Austrian lake surface temperatures and their relationship to regional air temperature and the North Atlantic Oscillation. *Limnol. Oceanogr.* **2001**, *46*, 1220–1227. [CrossRef]
47. Kheyrollah Pour, H.; Duguay, C.R.; Martynov, A.; Brown, L.C. Simulation of surface temperature and ice cover of large northern lakes with 1-D models: A comparison with MODIS satellite data and in situ measurements. *Tellus A Dyn. Meteorol. Oceanogr.* **2012**, *64*, 17614. [CrossRef]
48. Woolway, R.I.; Merchant, C.J. Amplified surface temperature response of cold, deep lakes to inter-annual air temperature variability. *Sci. Rep.* **2017**, *7*, 4130. [CrossRef] [PubMed]
49. Deyu, W.; Xuezhi, F.; Liguo, Z.; Jingyan, H.A.O.; Xiaoxiong, X.U. Relationship between blue algal bloom and water temperature in Lake Taihu based on MODIS. *J. Lake Sci.* **2008**, *20*, 173–178. [CrossRef]
50. Wu, H.R.; Zhang, X.T.; Liang, S.L.; Yang, H.; Zhou, G.Q. Estimation of clear-sky land surface longwave radiation from MODIS data products by merging multiple models. *J. Geophys. Res. Atmos.* **2012**, *117*, D22107. [CrossRef]
51. Yao, Y.J.; Liang, S.L.; Li, X.L.; Hong, Y.; Fisher, J.B.; Zhang, N.N.; Chen, J.Q.; Cheng, J.; Zhao, S.H.; Zhang, X.T.; et al. Bayesian multimodel estimation of global terrestrial latent heat flux from eddy covariance, meteorological, and satellite observations. *J. Geophys. Res. Atmos.* **2014**, *119*, 4521–4545. [CrossRef]
52. MPICH. Available online: https://www.mpich.org/ (accessed on 24 February 2017).
53. Brawer, B. *Introduction to Parallel Programming*; Academic Press Professional Inc.: San Diego, CA, USA, 1989; p. 422.
54. Windows Server. Available online: https://docs.microsoft.com/zh-cn/windows-server/windows-server (accessed on 1 December 2017).
55. Samba. Available online: https://www.samba.org/ (accessed on 1 December 2017).