Computing and Informatics, Vol. 33, 2014, 197–217

PARALLEL PROCESSING OF MASSIVE REMOTE SENSING IMAGES IN A GPU ARCHITECTURE

Peng Liu, Tao Yuan, Yan Ma*, Lizhe Wang
Dingsheng Liu, Shasha Yue

Institute of Remote Sensing and Digital Earth
Chinese Academy of Sciences
No. 9 Dengzhuang South Road, Haidian District
Beijing 100094, P. R. China
e-mail: `YanMa@ceode.ac.cn`

Joanna Kołodziej

Institute of Computer Science
Cracow University of Technology
Warszawska 24
31-155 Cracow, Poland
e-mail: `jokolodziej@pk.edu.pl`

Abstract. Profiting from the development of space remote sensing technology, the amount of remote sensing image data obtained by satellite is increasing dramatically; however, how to deal with these data quickly and efficiently has turned out to be a great computational challenge. With the rapid development of general-purpose GPU computing technology, researchers improved remote sensing applications based on GPU, and obtained good speedup. However, the current GPU parallel processes are not well adapted to the remote sensing image processing; furthermore, they have data loading, storage, and I/O problems. To solve these bottlenecks, this paper proposes three corresponding optimization strategies, and their effectiveness is confirmed by further experiments.

Keywords: GPU, remote sensing image processing, data intensive computing

* corresponding author

1 INTRODUCTION

Modern remote sensing technology has entered a new period that can dynamically and quickly provide a variety of earth observation data; as a result, the amount of data obtained by remote sensing is growing rapidly [14]. Furthermore, remote sensing image processing algorithms are complex and computation-intensive; thus, methods of quickly and efficiently dealing with the processing of remote sensing images represent one target of study [15, 16, 17].

Graphic process units (GPUs) are part of the rapid development of general computing architecture, and make calculations to improve performance. Today, the processing power of GPUs in numerical calculation is higher than that of CPUs, especially as GPUs' programmability, parallel processing ability, and scope of application are continuously being improved and expanded. Thus, they have become one important component by which a computer system can provide high performance ability. GPUs, as general-purpose, large-scale parallel processors, have the characteristics of strong floating point calculation, highly intensive computation, small volume, and a good performance power consumption ratio and performance-cost ratio. As a result, they provide a new method of solving the problems faced in remote sensing image processing, such as real-time insufficiency, low computing density, etc.

In recent years, domestic and foreign research institutions have been doing rich, extensive research in GPU-based remote sensing image processing algorithms and application systems. These studies have shown that a majority of remote sensing image processing algorithms exhibit good adaptability to GPU computational features, and make a better foundation for GPU accelerating remote sensing image processing. However, current GPU parallel processes are not well adapted to remote sensing image processing; furthermore, they have data loading, storage, and I/O problems. To solve these bottlenecks, this paper proposes three corresponding optimization strategies, and their effectiveness is confirmed by further experiments.

2 RELATED WORK

GPU computing is the use of a graphics processing unit (GPU) as a co-processor to accelerate CPUs for general-purpose scientific and engineering computing. A CPU consists of four to eight CPU cores, while a GPU consists of hundreds of smaller cores. This massively parallel architecture is what gives the GPU its high computing performance, and has obvious advantages over CPUs in terms of processing ability.

In recent years, domestic and foreign research institutions have carried out a variety of studies on the remote sensing image GPU parallel processing algorithms, various GPU processing algorithm libraries, and GPU applications or systems, and have obtained fruitful results.

Using GPU to accelerate processing for all kinds of remote sensing, image processing algorithms have resulted in greater related research achievements. The German Aerospace Center applied the technology to remote sensing data processing algorithms such as orthorectification and achieved real-time processing of aerial data from aviation aircraft [1]. Ashutosh Gupta used this technology to generate a GPU-based image matching method, and greatly improved the computational performance of hierarchical image matching techniques in the digital elevation model (DEM) generation algorithm [2]. These studies on GPU using the CUDA programming model employ data partitioning and thread assignment, so that each thread is responsible for processing one of the data blocks; thus, all threads together complete remote sensing image processing as a whole through labor division and cooperation.

There are many other similar studies. For example, there has been GPU implementation of the pixel purity index algorithm for hyperspectral image analysis [3], IHS transform fusion of remote sensing image data based on GPU [4], parallel implementation of nearest neighbor analysis based on GPGPU [5], parallel implementation of endmember extraction algorithms [6, 7], GPU implementation of target and anomaly detection algorithms [8, 9], parallel processing for normal mixture models of hyperspectral data [10], and neural signal processing [13].

The Tech-X company, under the support of the NASA SBIR scheme, has researched and developed the GPULib library [11]. Its design goal is to provide mapping from the application to GPU architecture for remote sensing image processing, medical image processing, and so on, without too much concern for the GPU hardware programming. Up to the present, GPULib has been used by Leiden University, NASA, US AFRL, the Laboratory of Atmospheric and Space Physics (LASP), and other organizations.

Jacket Engine is a GPU-based computing engine developed by the AccelerEyes Company specifically for MATLAB. Furthermore, Samuel Rosario-Torres has engaged in GPU-accelerated research based on the hyperspectral image analysis toolbox (HIAT) [12].

3 PROBLEM DEFINITION

Remote sensing has become the primary means of obtaining global spatial information, and has the characteristics of wide coverage, the ability to process large amounts of data, and the ability to access a wide variety of sources. There are various remote sensing image processing algorithms, but from the perspective of data processing, these can be divided into the following three major categories: point operations, local operations (geometric transform can be seen as a special local operation), and global operations. From the viewpoint of image processing, most algorithms can also be seen as a process of generating an image from another image, and the processed data for the most part are rules and local pixel sets.

3.1 Typical Remote Sensing Image Processing Algorithms in a GPU Architecture

3.1.1 Point-Operation Based Algorithms

Point operations are the simplest kind of image processing algorithms; it is only necessary to input the image pixel value as a parameter for simple single parameter function calculation, producing an output image corresponding to the pixel. In the calculation of the process, access to data from other pixels is not required; this calculation can be described as $g(x, y) = H[f(x, y)]$ (see also Figure 3).

$f(x, y)$ is a pixel gray value of the input image, $g(x, y)$ is a pixel gray value of the output image, and H is the gray transform function. For all the pixels of the whole image, the above equation can be repeatedly used to obtain the new output image. Moreover, with the multi-band characteristics of remote sensing images, this kind of point operation can be extended to multiple images or multi-band image processing, for example, composite operations of corresponding points among multiple bands of an image. In many image processing algorithms, point operations are most beneficial for GPU parallelization due to their single function and repeatable calculation. This image point operation represents a large proportion of remote sensing image processing. According to whether the calculation requires multiple bands to be involved in computing at the same time, point operations can be further divided into band-unrelated and band-related point operations. Typical algorithms of the former type are, for example, image reversion, logarithmic transformation, power transformation, threshold segmentation, etc. Typical algorithms of the latter type are NDVI calculation, water extraction, the drought model, image subtraction, image processing on average, and so on. In this paper, to avoid losing either type of algorithm, one of each kind is selected, specifically logarithmic transformation and NDVI calculation.

Logarithmic transformation is the basic image gray transform algorithm, and its general expression is:

$$g(x, y) = c \times \log(1 + f(x, y)).$$

Here, c is a constant, $f(x, y)$ is the original image pixel value, and $g(x, y)$ is the resulting image pixel value. Logarithmic transformation causes a narrow band and a low gray-level value in the input image to be mapped to a broadband output value. In image enhancement, it is often used to expand the dark pixels in the high-value compressed image. Figure 1 shows the GPU parallel procedure. Because the values after logarithmic transformation are floating point numbers, and not between 0 and 255, they require linear stretching.

The vegetation index represents the spectral values of the surface vegetation condition extracted from the multi-spectral remote sensing image data, and can reflect the status of plants in values. At present, the normalized difference vegetation index (NDVI) is most widely used. This index is more sensitive to changes in

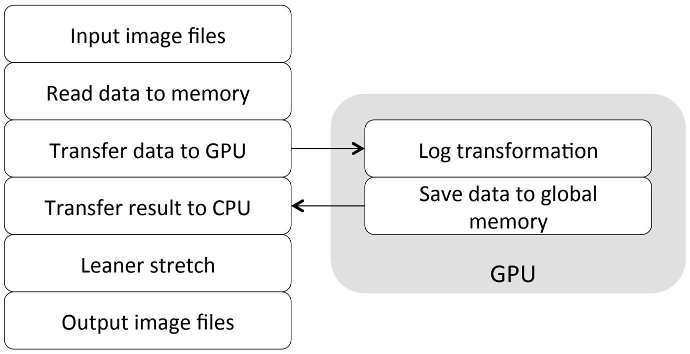


Figure 1. Log transform procedure

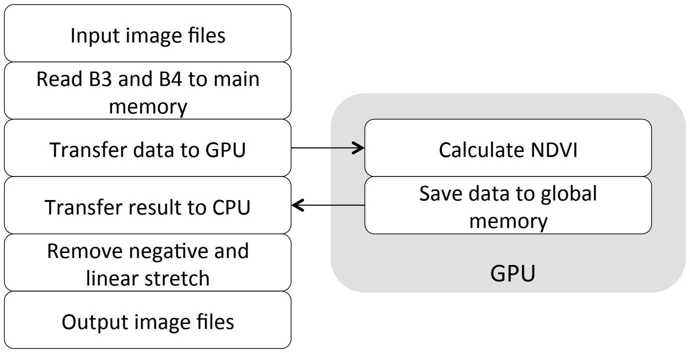


Figure 2. NDVI calculation procedure

soil background, and weakens atmospheric disturbances, thus greatly expanding testing sensitivity for vegetation coverage. The normalized difference vegetation index calculation formula for LANDSAT TM data is as follows:

$$NDVI = (B4 - B3)/(B4 + B3), \tag{1}$$

where $B3$ and $B4$ are the third and fourth band, respectively, of the TM data. The NDVI values range from -1 to 1, and can be very good when it comes to distinguishing non-vegetation and vegetation zones. An area is not a vegetation zone if the NDVI value is less than or equal to 0; otherwise, it is a vegetation zone, and the greater the value, the more dense the vegetation.

The GPU parallel procedure is shown in Figure 2. The NDVI value is a floating point between -1 and 1, but the NDVI values of vegetation area are greater than 0, so we need to remove the negative values and perform linear stretching. In the GPU parallel processing flow, each thread gives the NDVI calculation of one pixel in the output image.

3.1.2 Local-Operation-Based Algorithm

Local operations (also known as neighborhood operations), generally implemented by convolution operations, are widely used in image smoothing, sharpening, edge detection, and feature extraction. As a pixel gray value of the output image requires the neighborhood of the corresponding pixel gray value of the input image for the calculation, this operation can be described as $g(x,y) = H\left[\sum_{(s,t)\in S_{xy}} f(s,t)\right]$ (see also Figure 4).

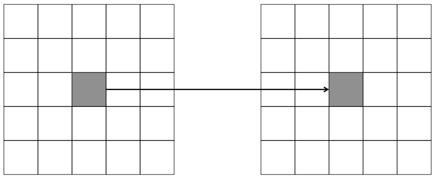


Figure 3. Point operation

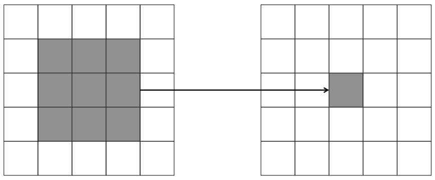


Figure 4. Local operation

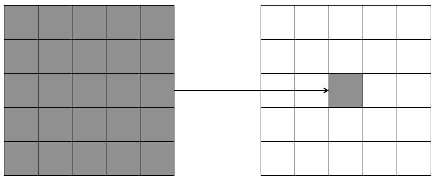


Figure 5. Global operation

S_{xy} is a pixel neighborhood of the input image, $g(x,y)$ is a pixel gray value of the output image, and H is the gray transform function. This class of algorithms has one of the most important features: One pixel gray value of the output image is related to one neighborhood of the input image, and the processes of calculating each pixel of the output image are independent of one another. Here, local operations show a certain correlation of input image data, but the specific calculation process, or application, is unrelated. For each pixel of the output image, the calculation is repeated, and the input image data are used to obtain all pixel values of the output images. Since the neighborhood of the input image is local and the pixels involved in the calculation are limited, this feature can also realize GPU parallel processing. At the same time, due to the increase of each pixel computation quantity, it is expected that a higher speed will be obtained under the premise of the input image data scheduling optimization.

In the remote sensing image processing algorithm, local operation algorithms represent most of the calculations. According to the different coordinate mapping between the input and output pixel points, the Gauss filter and image rotation were chosen as the typical algorithms. Compared to the Gauss filter, due to geometric deformation, the correspondence between the input and output pixels of image rotation is more complex.

The Gauss filter is used more often in the multi-scale analysis of remote sensing images, and is commonly used to establish the Gauss pyramid. The Gauss filter on the image generates image convolution using a convolution kernel consisting of the Gauss distribution function. Here, Gauss convolution kernel calculation formulas include Equation (2), where δ is used for Gauss standard deviation of normal distribution, and N is a convolution kernel size. The calculated Gauss convolution kernel also needs to be normalized, so the sum of all coefficients of the Gauss convolution kernel is 1. Typically, $\delta = 0.5$.

$$G(x, y, \delta) = \frac{1}{2\pi\delta^2} e^{-\frac{(x-\lfloor N/2 \rfloor)^2 + (y-\lfloor N/2 \rfloor)^2}{2\delta^2}} \tag{2}$$

According to probability theory, the convolution size should be selected for data in the range of $[-3\delta, 3\delta]$. The calculation formula with the standard deviation is as follows:

$$N = 1 + 2 \times \lceil 3 \times \delta \rceil.$$

The GPU parallel procedure for the Gauss filter is shown in Figure 6. In this process, each thread is responsible for calculating the Gauss filter for one pixel in the output image.

Geometric transformation can be seen as a special local operation, because an important step in geometric transformation is gray resample. This means that each pixel gray value of the output image is related to the neighborhood of one pixel of the input image. However, the difference from normal local operations is that the coordinate transformation relationship between the pixel of the input image and the neighborhood of the output image is more complex, and the coordinate difference between them is often large. This requires the input image local data scheduling to involve more sophisticated optimization in order to reach the higher speedup.

The image rotation algorithm is a kind of geometric transformation algorithm that is applied to rotate the image to an angle. The inverse operation method is used to compute the result image. The coordinates of the original image are obtained by counter-deduction from the coordinates of the result image. The bilinear interpolation algorithm is used to obtain the pixel values in the original coordinates. Firstly, the point on the resulting image is converted to a Cartesian coordinate system and then rotated clockwise to a certain angle. Finally, the rotated point is converted to the original coordinate.

The GPU image rotation procedure is shown in Figure 7. Every thread is responsible for one point's coordinate transform and bilinear interpolation.

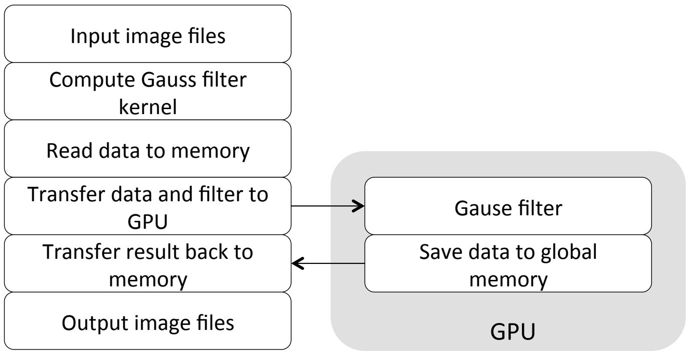

Input image files
Compute Gauss filter kernel
Read data to memory
Transfer data and filter to GPU
Transfer result back to memory
Output image files
Gause filter
Save data to global memory
GPU

Figure 6. Gauss filtering procedure

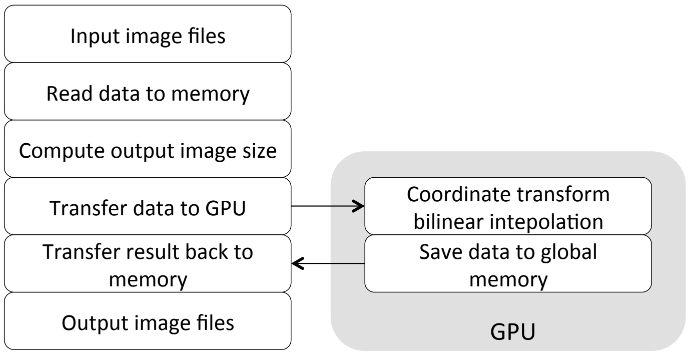

Input image files
Read data to memory
Compute output image size
Transfer data to GPU
Transfer result back to memory
Output image files
Coordinate transform bilinear intepolation
Save data to global memory
GPU

Figure 7. GPU image rotation procedure

3.1.3 Global-Operation-Based Algorithm

Global operations are the most complex kind of pixel level image processing. It is usually not possible to limit the amount of data required for each pixel gray value calculation of the output image; in some cases, this may be associated with the information for the whole image. All the pixels of the input image may be involved in the process of calculation, as shown in Figure 5. Typical algorithms include wavelet transform and Fourier transform.

The Fourier transformation process provided in the CUDA library has achieved a special optimization and maintains preferable speedup. The GPU procedure for Fourier transform is shown in Figure 8.

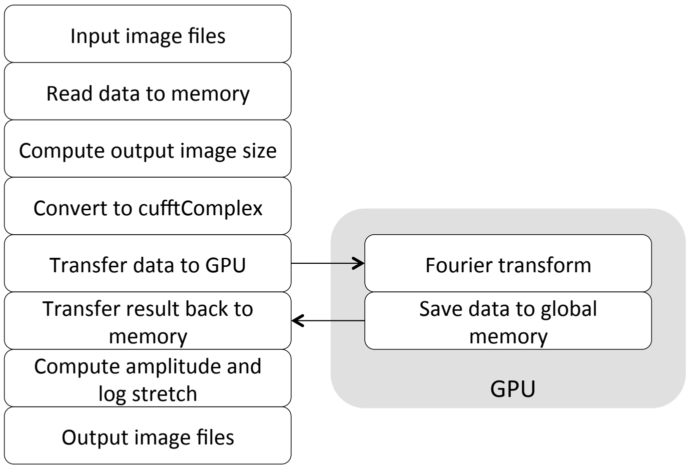


Figure 8. Fourier transform procedure

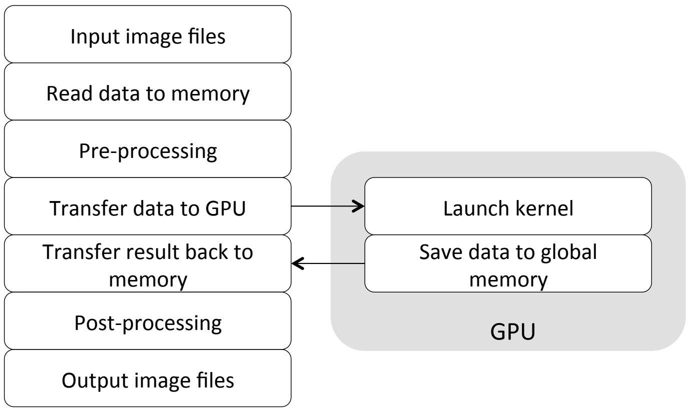


Figure 9. Parallel remote sensing image processing

3.2 Performance Bottleneck

The GPU parallel processes in remote sensing image processing are mostly transformed from GPU parallel processing in the general computing field, and the general parallel process is as shown in Figure 9. First, the remote sensing image data are loaded into the memory, and then the desired image data and processing parameters in the computation are transmitted to the GPU memory. After data partition and thread allocation, the kernel function for multi-thread parallel processing is initiated. After the completion of the processing, the resulting image data are transmitted back to the memory from the GPU memory; finally, they are written to the image file.

The current GPU parallel process has exhibited very good acceleration in speeding up the remote sensing image processing algorithms, but analyzing the current parallel process in combination with the characteristics of remote sensing image processing and GPU hardware architecture, the following problems and performance bottlenecks are observed.

3.3 Data Loading

The current GPU parallel processing procedure generally requires data to be loaded to the GPU memory all at once, but the GPU memory capacity is limited – for example, the Tesla GPU C1060's memory capacity is 4 GB. Therefore, the amount of data loaded to the GPU memory cannot be greater than the memory capacity. Otherwise, the original remote sensing image data need to be divided and combined, and this can be accomplished after multiple GPU parallel processes. The current GPU parallel process does not propose a proper data partitioning and combination strategy for remote sensing image data characteristics to solve the problem.

3.4 Data Storage

The existing GPU parallel process generally stores the image data in the global memory, but this is not cached and its access delay is long. Moreover, remote sensing image processing algorithms often need to obtain pixel values in a two-dimensional neighborhood, cannot meet the global memory coalesced access requirement, and only serial access sequentially; this results in a serious time consumption problem. The current GPU parallel process does not put forward a reasonable data storage strategy for the six different types of memory according to the features of the remote sensing image data and algorithms.

3.5 Data I/O

The present GPU parallel process needs to transmit data back and forth between the GPU memory and the host memory. The GPU memory and host memory are connected by the PCI-E bus, the bandwidth of which is bidirectional 8 GB/s in theory, far less than the bandwidth between global memory and on-chip memory. When the amount of remote sensing image data is large, data I/O time consumed between GPU memory and host memory cannot be ignored, and becomes a serious overall speed ascent bottleneck. The CPU and GPU in current parallel process work serially with an unbalanced load, and do not make full use of computing resources.

4 DESIGN AND IMPLEMENTATION

In order to analyze the bottleneck problem related to performance in the existing GPU, the total elapsed *serial time* and *parallel time* are decomposed in Equations (3)–(5).

$$\begin{aligned} \text{serial time} &= \text{CPU time} + \text{disk I/O} + \text{system overhead} && (3)\\ \text{parallel time} &= \text{GPU time} + \text{disk I/O} + \text{system overhead} && (4)\\ \text{GPU time} &= \text{compute time} + \text{data transfer time} + \text{GPU system overhead} && (5) \end{aligned}$$

Serial time is composed of CPU time, data I/O time, and system overhead time. Parallel time is composed of CPU time, data I/O time, and system overhead time, meanwhile CPU time is composed of CPU computing time, data transfer time, and GPU system overhead time. In the ideal situation, the data I/O time and system overhead time of the serial algorithm will be the same as those of the GPU parallel algorithm. The GPU parallel algorithm only speeds up in computing time, which becomes GPU computing time. The speedup occurs when the GPU computing time is less than the CPU computing time. The data I/O time, system overhead time, data transfer time, and GPU system overhead time are never able to come down. Only overlapping this time through system scheduling can weaken its impact on program performance. Only GPU computing can be optimized.

4.1 Design of the Optimization Strategy

Firstly, to divide tasks for data to be processed, it is necessary that the different tasks be unrelated and able to be executed in parallel. Then, three kinds of threads need to be started, specifically the data input thread, data processing thread, and data output thread. There can be one or more of each kind of thread.

The data input thread obtains data entry tasks from the task list; it is responsible for reading data from the input remote sensing image file and putting them into the data input buffer queue. The data processing thread receives the data processing task from the data input buffer queue, transmits the data and processing parameters to the GPU memory, binds the data to the texture memory, and starts the kernel for multi-threaded parallel processing. It then transforms the resulting data back to the host memory after the processing is completed, and then puts them into the data output buffer queue. The data output thread receives data output tasks from the data output buffer queue, which is responsible for writing the processed resulting data to the output remote sensing image file.

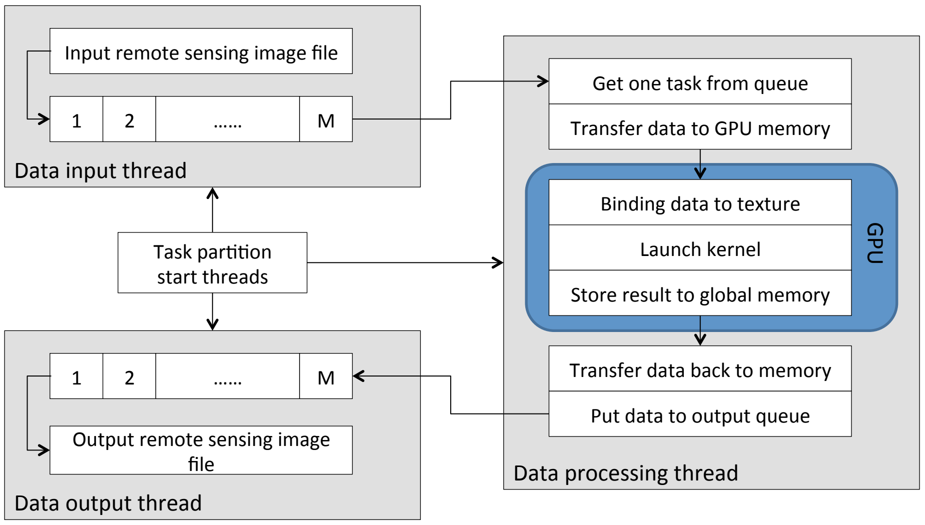

Input remote sensing image file
1
2
......
M
Data input thread
Task partition start threads
1
2
......
M
Output remote sensing image file
Data output thread
Get one task from queue
Transfer data to GPU memory
Binding data to texture
Launch kernel
Store result to global memory
GPU
Transfer data back to memory
Put data to output queue
Data processing thread

Figure 10. Optimized remote sensing image GPU parallel processing

Three optimization strategies are adopted to resolve the performance problems independently.

4.1.1 Data Loading Strategy

The GPU memory capacity is 4 GB, which is sufficient for most remote sensing image processing. To process remote sensing image data greater than the memory capacity, task partitioning and data block processing need to be performed. Because GPU acceleration is associated with the amount of data, the data block cannot be too small, and should be as close as possible to the memory capacity. The data buffer technology can be used to restrict the amount of data loaded at one time.

4.1.2 Memory Access Optimization Strategy

The texture memory stores image data, and can improve the algorithm of random data access or two-dimensional data access efficiency. The linear filter model for remote sensing image processing often requires a resampling operation – the texture memory does so conveniently, and can greatly improve the performance of the program. A constant memory suitable for the storage of remote sensing image processing algorithms requires frequent access read-only parameters, such as default parameters, a weight coefficient matrix array, etc. The global memory is fit for storing remote sensing image results data. Because the texture memory is read-only memory, processed remote sensing image data still cannot be stored in the texture memory; rather, it can only be stored in the global memory. Shared memory is suitable for storing the data sharing between multiple threads in remote sensing image process algorithms, such as the parameters of the algorithm and filter array to which each thread will need access. In addition, if the local variable is too great, or the register is used in excess, then the shared memory can be used as registers, although in this case computing time is slower than when registers are used but faster than local memory.

4.1.3 I/O Hiding Strategy

Data buffer technology allows for remote sensing image file read and write operations and GPU processing step decoupling; the use of multi-threading technology can carry out remote sensing image file read and write GPU image processing at the same time, as well as improving the utilization rate of the resources so as to achieve load balancing, thereby hiding the data transmission time in the process performance.

As shown in Figure 11, this processing flow decouples the data I/O and the data process task through the data input and output buffer, uses multi-threading technology to let the CPU and GPU work at the same time, eliminates the CPU and GPU load imbalance that occurs in the traditional process, and improves the utilization of computing resources.

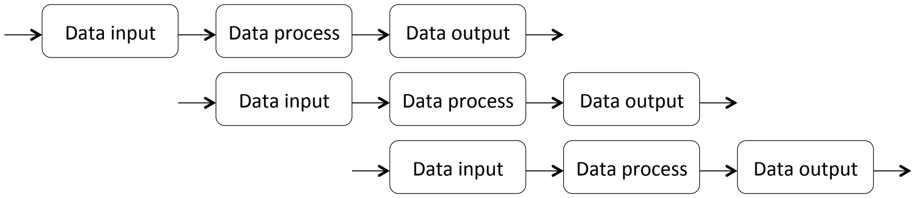


Figure 11. I/O hiding by multi-thread technology

4.2 Implementation of the Optimization Strategy

All of the five typical algorithms discussed above can use a data loading strategy to limit the amount of remote sensing image data loaded into the memory and computed by the GPU. This method can also effectively deal with remote sensing images that exceed the storage capacity. The storage optimization strategy takes advantage of texture speedup to bind the remote sensing image date with the texture memory of GPU. Data I/O and data processing take place at the same time through the I/O hidden strategy. Thus, the operation impairs the influence of data I/O on program performance.

Because the logarithmic transformation algorithm and NDVI calculation have no processing parameters, there is no need to optimize storage access after binding the remote sensing image data to the texture memory. The processing parameters of the Gauss filter algorithm represent the Gauss convolution kernel. This can be moved into shared memory to optimize performance. Bilinear interpolation in the image rotation algorithm can take advantage of the GPU's texture filtering mode to optimize performance.

5 EXPERIMENTS AND PERFORMANCE EVALUATION

In the three strategies, the algorithm to enhance performance strategy uses only the memory optimization strategy and I/O strategy; the data loading strategy is used when the quantity of remote sensing data is too large to be processed and put forward, and has no effect on the performance of the program.

Five single-band remote sensing images numbered 1–5 are selected to implement the logarithmic transformation algorithm in order to compare the performance of CPU multi-core parallel computing and GPU multi-core parallel computing. The sizes of these images are $4\,096 \times 4\,096$, $4\,096 \times 8\,192$, $8\,192 \times 8\,192$, $8\,192 \times 16\,384$, and $16\,384 \times 16\,384$. The acceleration ratio is the ratio of the total multi-core CPU parallel computing time to the total multi-core GPU parallel computing time. The total elapsed time includes data I/O time, system overhead time, and so on.

Device	Specifications
CPU	1 × Intel(R) Xeon(R) E5504 CPU 4 Cores, 2.00 GHz, L3 Cache 4 MB
Memory	4×2 GB ECC DDR3-1066
Disk	1×1 TB SATA
Network	2 × 1 000 M Adaptive Ethernet Card
GPU	NVIDIA Tesla C1060
Compilers	GCC 4.1.2, NVCC 4.0
CUDA	4.0

Figures 12 and Figure 13, respectively, are the NDVI calculation and the logarithmic transform in remote sensing image data using a global memory and texture memory with the acceleration of contrast. As can be seen, almost no changes in acceleration ratio occur. Since the point of operation class algorithm input image each pixel is only read once, it is unable to get the texture memory cache to accelerate. Thus, for the computation of the algorithm, the texture memory optimization strategy failed to demonstrate an accelerating effect.

Figures 14 and 15, respectively, show the Gauss filtering and image rotation algorithm in the global memory and texture memory with the acceleration of contrast. As can be seen, with the increase in data quantity, the speedup increase

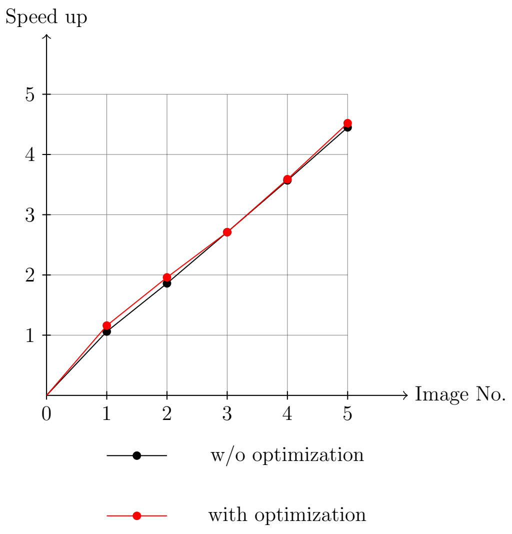


Figure 12. Log transform texture optimization

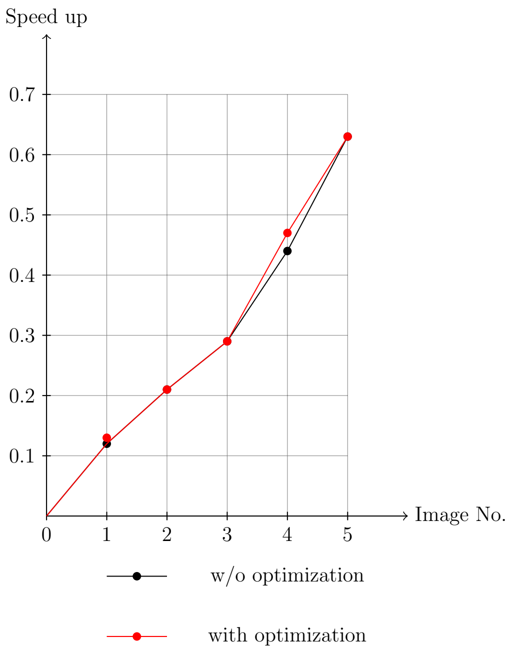


Figure 13. NDVI calculation texture optimization

continuously improves, the Gauss filter can obtain a 0.2 times greater acceleration ratio increment, and an image rotation of 1.31 times can be obtained.

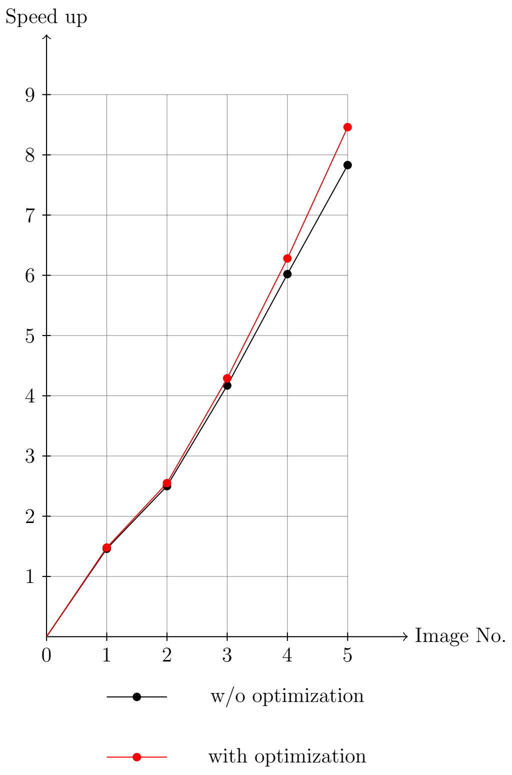


Figure 14. Gauss filter texture optimization

Because each Gauss filtering needs to read the input image of a neighborhood of pixels, each pixel will be read repeatedly; a texture memory cache can be used to accelerate the reading. In addition to using the texture cache, image rotation can also make use of its filtering mode, without the need for additional bilinear interpolation calculation, thereby saving a lot of time. Therefore, the local computing algorithms and texture memory optimization strategy can play a very good role in the acceleration effect.

Figure 16 is the Gauss filter algorithm of the Gauss convolution kernel in the global memory and shared memory under accelerated contrast. As can be seen, with the increase of data quantity and accelerated growth in relation to the larger amplitude, the maximum amplitude can reach 1.15. Because each Gauss convolution kernel's thread calculations require the use of values stored in the shared memory, the access speed is almost as fast as the register, and this can greatly improve process performance. Therefore, for remote sensing image processing algorithms that require

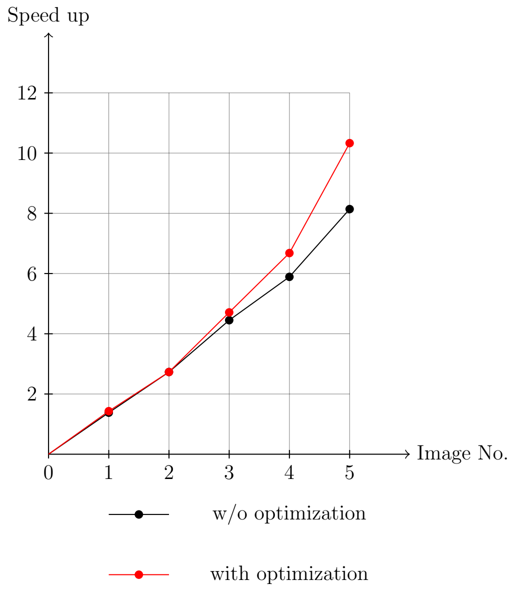


Figure 15. Image rotation texture optimization

frequent access algorithm parameters, a filter array placed in the shared memory has an obvious performance advantage.

Figure 17 shows a logarithmic transformation algorithm where the data size is 8 192×8 192 and the band numbers are 1, 2, 4, 8, and 16, where the I/O hiding strategy is employed and the contrast is accelerated. As can be seen, speedup improves, and the highest result is 0.17 times the acceleration ratio amplitude. The data buffer technique can effectively control the memory data size, so that data can be treated effectively. The use of multi-threading technology makes the logarithm transform the parallel algorithm of the disk I/O and GPU data processing at the same time in order to be able to hide a portion of the I/O time.

6 CONCLUSION

This article analyzed and classified data processing in remote sensing image data and processing algorithms, as well as the combination of GPU structure characteristics. The corresponding treatment process was delineated, focusing on the point of computing, local operations, and global operation of three kinds of typical algorithm parallelization and optimization problems. In view of the existing process in the presence of data loading, data storage, and the data I/O problem, corresponding strategies – specifically the data loading strategy, memory optimization strategy, and I/O strategy – were put forward. It was further confirmed in experiments that the optimization strategy is effective.

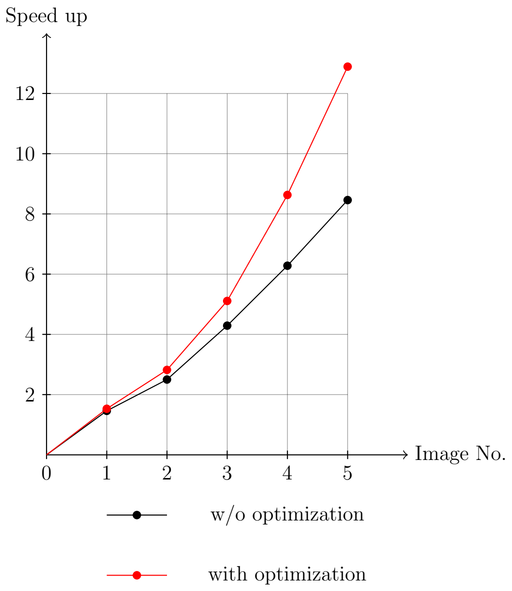


Figure 16. Gauss filter shared memory optimization

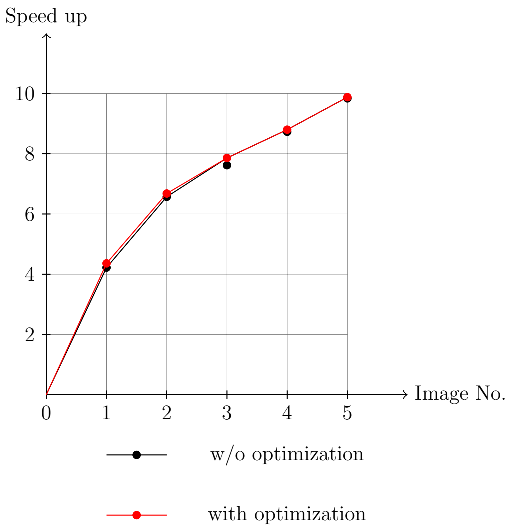


Figure 17. Log transform I/O hiding optimization

REFERENCES

[1] Thomas, U.—Kurz, F.—Rosenbaum, D.—Mueller, R.—Reinartz, P.: GPU-Base Orthorectification of Digital Airborne Camera Images in Real Time. Proc. of The Int'l Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. XXXVII. Part B1, Beijing 2008, pp. 589–594.

[2] Gupta, A.—Devakanth Naidu, S.—Srinivasan, T. P.—Gopala Krishna, B.: A GPU Based Image Matching Approach for DEM Generation Using Stereo Imagery. In Proc. of Nirma Univ. Int'l Conf. on Engineering (NUiCONE) 2011, pp. 1–5.

[3] Paz, A.—Plaza, A.: Cluster Versus GPU Implementation of an Orthogonal Target Detection Algorithm for Remotely Sensed Hyperspectral Images. In Proc. of the CLUSTER 2010, pp. 227–234.

[4] Lu, J.—Zhang, B.: An Accelerated IHS Transform Fusion of Remote Sensing Image Data Based on GPU. ESIAT, Vol. 1, 2009, pp. 492–496.

[5] Zhao, Y.—Chen, B.—Fang, Y.—Huang, Z.—Liu, Y.—Yu, H.: A Parallel Implementation of Nearest Neighbor Analysis Based on GPGPU. In Proc. of the 19th Int'l Conf. on Geoinformatics 2011, pp. 1–6.

[6] Plaza, A.—Plaza, J.—Sanchez, S.: Parallel Implementation of Endmember Extraction Algorithms Using NVidia Graphical Processing Units. In Proc. of the Geoscience and Remote Sensing Symposium – IGARSS '09, pp. 208–211.

[7] Setoain, J.—Prieto, M.—Tenllado, C.—Plaza, A.—Tirado, F.: Parallel Morphological Endmember Extraction Using Commodity Graphics Hardware. IEEE Geoscience and Remote Sensing Letters, Vol. 4, 2007, No. 3, pp. 441–445.

[8] Paz, A.—Plaza, A.: Clusters Versus GPUs for Parallel Target and Anomaly Detection in Hyperspectral Images. In Proc. of EURASIP 2010.

[9] Setoain, J.—Tenllado, C.—Prieto, M.—Valencia, D.—Plaza, A.—Plaza, J.: Parallel Hyperspectral Image Processing on Commodity Graphics Hardware. In: Proc. of the ICPP Workshop 2006, pp. 465–472.

[10] Blanco Heras, D.—Arguello, F.—Lopez Gomez, J.—Becerra, J.—Duro, R. J.: Towards Real-Time Hyperspectral Image Processing, a GPGPU Implementation of Target Identification. In Proc. of IDAACS 2011, pp. 316–321.

[11] Messmer, P.—Mullowney, P. J.—Granger, B. E.: GPULib: GPU Computing in High-Level Languages. Computing in Science and Engineering, Vol. 10, 2008, No. 5, pp. 70–73.

[12] Rosario-Torres, S.—Velez-Reyes, M.: Speeding Up the MATLAB Hyperspectral Image Analysis Toolbox Using GPUs and the Jacket Toolbox. In Proc. of the First Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing, 2009, pp. 1–4.

[13] Chen, D.—Wang, L.—Ouyang, G.—Li, X.: Massively Parallel Neural Signal Processing on a Many-Core Platform. Computing in Science and Engineering, Vol. 13, 2011, No. 6, pp. 42–51.

[14] Huang, F.—Liu, D.—Li, X.—Wang, L.—Xu, W.: Preliminary Study of a Cluster-Based Open-Source Parallel GIS Based on the GRASS GIS. Int. J. Digital Earth, Vol. 4, 2011, No. 5, pp. 402–420.

[15] Wang, L.—Chen, D.—Deng, Z.—Huang, F.: Large Scale Distributed Visualization on Computational Grids: A review. Computers & Electrical Engineering, Vol. 37, 2011, No. 4, pp. 403–416.

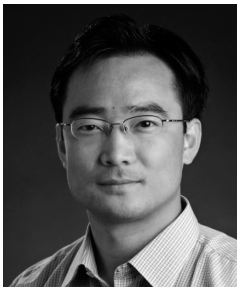
[16] Wang, L.—Kunze, M.—Tao, J.—von Laszewski, G.: Towards Building a Cloud for Scientific Applications. Advances in Engineering Software, Vol. 42, 2011, No. 9, pp. 714–722.

[17] Wang, L.—Fu, C.: Research Advances in Modern Cyberinfrastructure. New Generation Comput., Vol. 28, 2010, No. 2, pp. 111–112.

Peng Liu received the his M. Sc. degree in 2004 and the Ph. D. degree in 2009, both from Chinese Academic of Science. From 2009, he is an Assistant Professor at the Institute of Remote Sensing and Digital Earth, Chinese Academy of Sciences. From May 2012 to May 2013, he was with Department of Electrical and Computer Engineering, George Washington University as a Visiting Scholar. His research is focused on sparse representation, compressive sensing, image processing and their applications to remote sensing.

Tao Yuan is a master student at Center for Earth Observation and Digital Earth, Chinese Academy of Sciences.

Yan Ma is an Assistant professor at Center for Earth Observation and Digital Earth, Chinese Academy of Sciences. Her research interests include high performance computing and remote sensing image processing.



Lizhe Wang is a Full Professor at Center for Earth Observation and Digital Earth, Chinese Academy of Sciences, China. He also holds a ChuTian Scholar Chair Professor position at School of Computers, China University of Geosciences. He received his Bachelor and Master of Engineering degrees from Tsinghua University, China and Doctor of Engineering from University Karlsruhe, Germany. His research interests include high performance computing, data-intensive computing, and Grid/Cloud computing.

Dingsheng Liu is a Professor at Center for Earth Observation and Digital Earth, Chinese Academy of Sciences. His research interests include remote sensing image processing and parallel computing.

Shasha Yue is a doctoral student at Center for Earth Observation and Digital Earth, Chinese Academy of Sciences.

Joanna Kołodziej graduated in mathematics from the Jagiellonian University in Cracow in 1992, where she also received the Ph. D. in computer science in 2004. She joined the Department of Mathematics and Computer Science of the University of Bielsko-Biała as an Assistant Professor in 1997. She has been serving as PC Co-Chair, General Co-Chair and IPC member of several international conferences and workshops including PPSN 2010, ECMS 2011, CISIS 2011, 3PGCIC 2011, CISSE 2006, CEC 2008, IACS 2008–2009, ICAART 2009–2010. She is Managing Editor of IJSSC Journal and serves as a EB member and guest editor of several peer-reviewed international journals.