A Large-Batch Orthorectification Generation Method Based on Adaptive GPU Thread Parameters and Parallel Calculation

Ruyan Zhou, Shangcheng Hu, Zhonghua Hong, *Member, IEEE*, Xiaohua Tong, *Senior Member, IEEE*, Shijie Liu, *Member, IEEE*, Haiyan Pan, Yun Zhang, Yanling Han, Jing Wang, and Shuhu Yang

***Abstract*—Orthorectification reflects a large amount of real and objective information, such as the characteristics of images and the geometric accuracy of maps. Conducting a large batch of orthorectification is a process with high time cost owing to the pixelwise correction each image. A common approach is to use graphics processing unit (GPU) parallel computing to accelerate orthorectification processing. However, most of the existing GPU acceleration studies have adopted experimental testing methods to determine thread parameters, which are inapplicable to different GPUs and affect the GPU acceleration performance. We put forward an adaptive calculation method for GPU thread parameters based on the performance parameters of different GPUs and by simultaneously blocking the image automatically according to the GPU memory space. We used 112 ZY-3 images to test the adaptive GPU and compare it to a general GPU. The experimental results show the following: first, for a single ZY-3 image, the GPU acceleration by the adaptive calculation method presented in this article is 43.22% higher than that by the general GPU, and the correction time is 34.41 times faster than that of the central processing unit. The result of the automatic image blocking was the same as that of the artificial blocking. Second, the experimental performance on four different GPUs indicated that all GPUs exhibited a significant speed boost. Third, for large-batch images, the GPU acceleration by the adaptive GPU was 32.6% higher than that by the general GPU, which provides an adaptive optimization strategy for large-batch image orthorectification.**

***Index Terms*—Adaptive, graphics processing unit (GPU), large batch, orthorectification, thread parameters.**

I. Introduction

IN RECENT years, high-resolution remote sensing satellites, for example, IKONOS, QuickBird, WorldView, TianHui, ZiYuan, GaoFen, etc., have been launched worldwide to collect data with different repeat cycles and resolutions. Orthorectification is an image processing method that can obtain the correct geographical coordinates of RS images by eliminating the geometric distortions due to terrain and sensor errors [1]. The generated orthoimagery after correction contains substantial real and objective information, not only including the geometric accuracy of maps but also the characteristics of the images. Therefore, the rapid generation of large-scale digital orthophoto maps has garnered considerable attention to meet the requirements of different application scenarios, such as postdisaster monitoring and map drawing [2].

There are two methods of orthorectification: direct and indirect. Direct correction starts from the original satellite images and directly calculates the geodetic reference space coordinates corresponding to each pixel using the algorithm models. Indirect correction is based on simulative orthoimagery; the first affine transforms the image coordinates to ground coordinates. Subsequently, its coordinates on the original image are calculated using an algorithmic model, and finally, grey value resampling is conducted to generate the orthoimagery. Orthorectification algorithms include the physical sensor and general empirical models [3]. With the collinearity conditions, the former expounds a calculation formula of the original image coordinates and actual object geographic coordinates with strict physical parameters [4]. The ratio of one or two polynomials consists of a general empirical model, which is a mathematical fit of physical sensor models. The rational function model (RFM) is used as most general empirical model, and is expressed as transformation relation by the cubic polynomial ratio [5], [6], [7]. Tenuous level and smooth splines modeling technology is used to abate the deviation of the RFM polynomial coefficients for correction [8]. Oh and Jung [9] presented an efficient method that integrates time and cost for automatic rational polynomial coefficients (RPCs) offset compensation correction of high-resolution satellite images. A lookup table can reduce workload of polynomial coefficients calculation and accelerate the correction of images [10]. Wang et al. [11] computed the image orientation parameters based on the plane block adjustment method before correcting every satellite image by the RFM. However, orthorectification is a pixelwise process, which leads to a large number of calculations. Consequently, many existing research work is based on small-batch images, but the orthoimagery generation of large-batch satellite images is relatively less.

Manuscript received 10 November 2022; revised 3 February 2023 and 16 April 2023; accepted 9 May 2023. Date of publication 15 May 2023; date of current version 26 May 2023. This work was supported in part by the National Key R&D Program of China under Grant 2018YFB0505400 and in part by the National Natural Science Foundation of China under Grant 41871325. *(Corresponding author: Zhonghua Hong.)*

Ruyan Zhou, Shangcheng Hu, Zhonghua Hong, Haiyan Pan, Yun Zhang, Yanling Han, Jing Wang, and Shuhu Yang are with the College of Information Technology, Shanghai Ocean University, Shanghai 201306, China (e-mail: ryzhou@shou.edu.cn; m210911520@st.shou.edu.cn; zhhong@shou.edu.cn; hy-pan@shou.edu.cn; y-zhang@shou.edu.cn; ylhan@shou.edu.cn; wangjing@shou.edu.cn; shyang@shou.edu.cn).

Xiaohua Tong and Shijie Liu are with the College of Surveying and Geo-Informatics, Tongji University, Shanghai 200092, China (e-mail: xhtong@tongji.edu.cn; liusjtj@tongji.edu.cn).

Digital Object Identifier 10.1109/JSTARS.2023.3276219

High-performance computing (HPC) is used in various applications with the advantages of the acceleration aspects of large data volume computing [12], [13], [14]. The HPC model includes heterogeneous computer network, cloud, field programmable gate array, and cluster-based multi-central processing unit (CPU) and graphics processing unit (GPU) systems [15], [16], [17]. The high-efficiency processing algorithms are explored in remote sensing image with HPC, such as hyperspectral imaging and endmember extraction [18], [19], [20], [21]. They concluded that HPC technology exhibits a significant improvement in efficiency compared to traditional computing methods. Owing to the large amount of data from high-resolution satellite images, a CPU-based single processor system cannot satisfy the requirements of fast and real-time data processing, and the multi-CPU system and FPGA are limited by high cost and complex programming; thus, GPU parallel computing is a better choice to accelerate the orthorectification processing. A new inexact Newton beam adjustment algorithm based on a multicore CPU and a single GPU, and the results of that showed that the efficiency in speed of a single GPU increased by two- to three-fold compared to that of a multicore CPU [22]. GPU parallel computing is applied to accelerate the orthorectification process of hyperspectral and unmanned aerial vehicle (UAV) images [23], [24]. Dai and Yang [25] implemented delicate parallel resampling to achieve fast correction. The images can be corrected fast based on the flow model and inverse sensor algorithm in a GPU-accelerated cluster environment [26], and the efficiency of input–output (I/O) operation in GPU correction by maximizing the device occupancy, which enhances the memory access efficiency [27]. Multiple GPUs are applied to efficient processing of modulation transfer function compensating and geo-rectification, and further improved the GPU performance in three aspects: memory throughput, instruction control, and multistream processing [28]. This method artificially divides an image into multiple blocks, which is unsuitable for large high-resolution satellite image processing operations by different GPUs. Li et al. [29] used GPU to accelerate the correction processing of UAV image distortions by configuring the best grid and multilevel memory. However, the Compute Unified Device Architecture (CUDA) thread parameters are determined by experimental testing, which implies that this method may be unsuitable for applications on different GPUs.

According to the aforementioned analysis, some problems still exist in the current research: first, in most studies implementing the existing GPU accelerating correction, the thread block parameters are determined experimentally, which is inapplicable to different GPUs and affects the acceleration efficiency; Second, in the current methods, images are corrected and need to be artificially partitioned, which affects the automation of correction. Therefore, the purpose of this study is to propose an automatic and adaptive GPU thread parameter calculation method for the generation of large-scale orthoimagery to improve the calculation efficiency of orthorectification. Two critical problems are explored in this regard: the adaptive calculation of thread parameters in different GPUs; and the correction of the automatic blocking of the image.

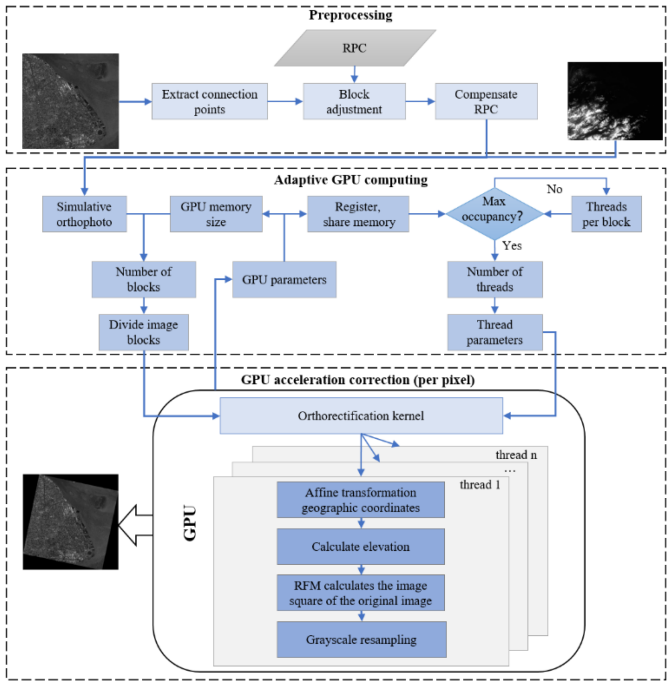


Fig. 1. Orthorectification flow of adaptive GPU calculation method.

The contributions of this article are as follows.

1) We utilize GPU parallel computing to accelerate the indirect orthorectification process; the proposed method is applied to large-area and large-batch high-resolution satellite images, which solves the low time efficiency problem of conventional orthorectification.
2) We propose an adaptive GPU thread parameter calculation method, which solves the problem of optimized calculation to determine the thread parameters of GPUs, and significantly decreases the time requirements.
3) We propose an automatic image blocking method to solve the problem of GPU-accelerated correction of large images that needs to be partitioned according to the size of the GPU memory.

The main organizational framework of this article is as follows: the adaptive GPU thread parameters, automatic blocking of the image method, and the specific process of orthorectification are explained in Section II. Section III provides the results and analysis of the orthorectification experiment. A discussion of the influence of thread parameters is presented in Section IV. Finally, Section V concludes this article.

II. Method

The process comprises three main steps, which are illustrated in Fig. 1. First, the satellite image is preprocessed, and the connection points are extracted for adjustment, and the result is used to compensate for the RPC. Second, we calculate the geographical longitude and latitude of satellite image four corner points to establish a simulative orthoimagery, and adaptively partition the image and calculate the thread parameters using the GPU. Finally, we use the GPU to calculate the original image

coordinates for each pixel along with the grey value resampling to generate the orthoimagery.

A. Preprocessing

Fig. 1 shows that the first step in this study is adjustment preprocessing, which eliminates the relative positioning error of the satellite images and the existing error in the initial RPC. First, The SIFT can detect points with the same characteristics and corresponding physical information in the image was used to extract the connection points from the images [30], and the least-squares method was used to reduce some useless connection points. The block adjustment model was constructed according to the connection points, and the common affine transformation model was used to construct the difference equation for the RFM. The object parameters and other information can be calculated using the forward intersection calculation method and iteratively solved to obtain the adjustment parameters [31], [32], [33]. Finally, the adjustment results are used to compensate for the RPC parameters and update the RPC files, which are convenient for subsequent calculations using the RFM.

B. Adaptive Calculation of the GPU Thread Parameters and Image Blocking

1) Automatic Image Blocking: When a GPU is used to accelerate the orthorectification of images, it is necessary to cut the images into several blocks when dealing with large satellite image data owing to the limited storage space of the GPU. The large image mentioned here is relative to the GPU memory space, and when the data size of the processed image exceeds the GPU memory space, it is considered as the large image. Generally, images are artificially divided into small blocks, which are unsuitable for different GPUs. Therefore, we propose an automatic image-blocking method based on the size of the remaining available video memory space and processing data size of the GPU. The data used are as follows: input data are the original image and the digital elevation model (DEM), whose sizes are $\mathrm{R_{img}}$ and $\mathrm{R_{DEM}}$, respectively; Output data are the orthoimagery, simulative orthoimagery are considered as orthoimagery after grayscale assignment, the size of the image is the ratio of maximum and minimum latitude and longitude differences to pixel width and height, the product of the data type of the image and the image size is the size of the memory occupied by the orthoimagery, which is $\mathrm{R_{DOM}}$; the aforementioned data are put into the GPU; and $\mathrm{R_{Gfree}}$ is the free space of the GPU memory. As the amount of data in DEM is small compared to that in the original images, it is unnecessary to partition the DEM. Thus, we only divide the input original image and the output orthoimagery, and the calculation scheme is as follows.

Case 1: The total size of the input and output data is less than the available space of the GPU memory, and no division is necessary.

Case 2: The sum of the input and output data is greater than the available space of the GPU memory; the original image and orthoimagery need to be partitioned, and the number of blocks is defined as $\mathrm{Block_n}$. The range of the orthoimagery block reflected in the original image is wider, and the excessive part size is $\mathrm{k} * R_{img}$; $k = (\mathrm{latitude}_{ul} - \mathrm{latitude}_{ur})/(\mathrm{latitude_{max}} - \mathrm{latitude_{min}})$, where $\mathrm{latitude}_{ul}$ and $\mathrm{latitude}_{ur}$ are the latitudes of the upper left and upper right corner points of the original image, respectively; and $\mathrm{latitude_{max}}$ and $\mathrm{latitude_{min}}$ are the maximum and minimum latitudes of the original image, respectively. Image block size was $(\frac{R_{\mathrm{img}}}{\mathrm{Block}_n} + k{*}R_{\mathrm{img}})$, and the orthoimagery block size was $\frac{R_{\mathrm{dom}}}{\mathrm{Block}_n}$ after blocking the images. The formula for the calculation of the image blocks is defined as follows, and ceil () is rounded up:

$$\begin{cases} \mathrm{Block}_n = 1 R_{\mathrm{total}} \le R_{G\mathrm{free}} \\ \mathrm{Block}_n = \mathrm{ceil}\left(\frac{R_{\mathrm{img}} + R_{\mathrm{DOM}}}{R_{G\mathrm{free}} - R_{\mathrm{DEM}} - k{*}R_{\mathrm{img}}}\right) R_{\mathrm{total}} > R_{G\mathrm{free}} \end{cases}. \tag{1}$$

2) Adaptive Calculation of GPU Thread Parameters: When using the CUDA framework, the pixelwise correction processing part with high parallelism is mapped into the kernel function of CUDA, and the kernel function that starts the correction must set the size of the thread blocks and grids. Conventional GPU acceleration sets the thread block size to 256, which is a two-dimensional size of 16 × 16. In previous studies, the size of the thread block was tested from 64 to 1024, and the optimal number of thread blocks was selected by testing the kernel function performance. The selected number of threads is not always the best for different GPUs, and unsuitable thread parameters will affect the running time of the kernel function. Therefore, we propose an adaptive GPU thread parameter calculation method to determine the thread parameters in CUDA according to the different performance parameters of the GPU to ensure that different GPUs can accelerate the correction process.

Fig. 2 shows that this method first obtains the physical performance parameters from the GPU; for example, the available registers and the available share memory (SM) size in a streaming multiprocessor. These parameters are then used to calculate the thread blocks per SM that can be limited by threads, registers, and SM, and calculate the number of warps. The minimum value of the three warp numbers was used to calculate the warp occupation rate and filter out the thread block size with the largest kernel occupation rate. If multiple thread blocks have the same occupation rate, the minimum value is considered as the final thread block size. The thread grid size and boundary qualifier parameter can then be determined according to the thread block size.

From the parameters of the GPU, we can obtain the number of threaded warps used by each SM, denoted as Num_{warp_SM}. A thread warp contains Num_{thread_warp} threads; the thread warp's allocation granularity is Num_{warp_alloc}; register allocation unit size is Num_{reg_alloc}; An SM has Num_{reg_SM} registers; and a thread has Num_{reg_thread} registers. Therefore, we can calculate the number of threaded warps in an SM limited by the registers

$$Num_{warp\_regs} = \mathrm{floor}\left(\frac{Num_{reg\_SM}}{\mathrm{ceiling}(Num_{reg\_thread} * Num_{thread\_warp},\ Num_{reg\_alloc})},\ Num_{warp\_alloc}\right). \tag{2}$$

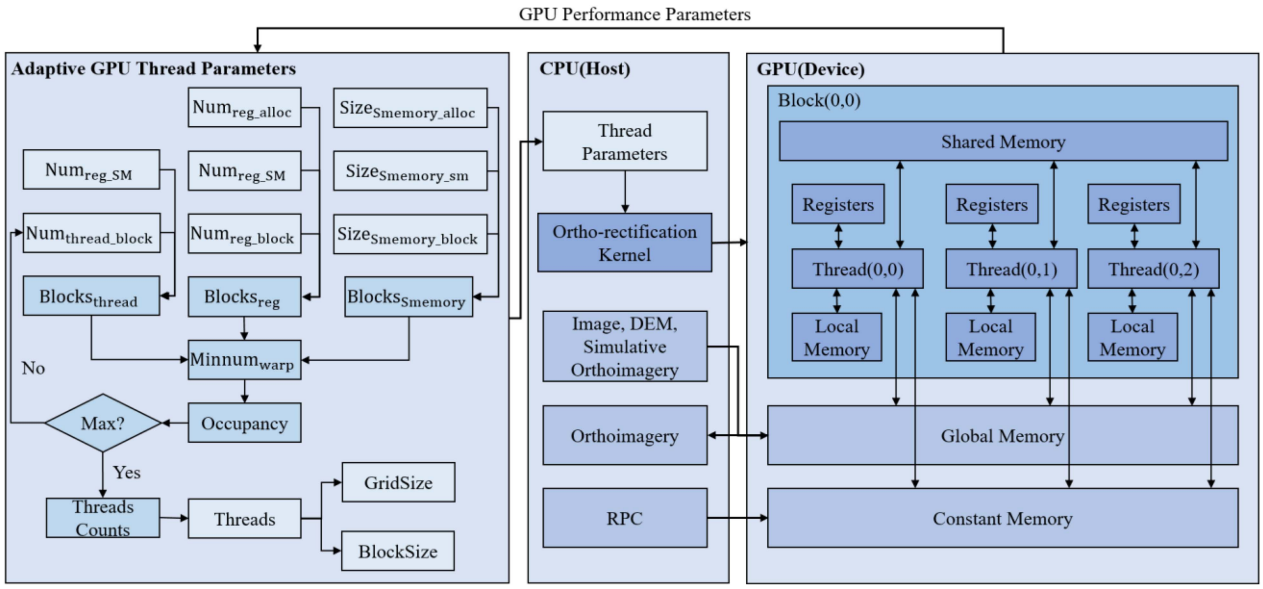


Fig. 2. Flowchart of the adaptive GPU thread parameter calculation.

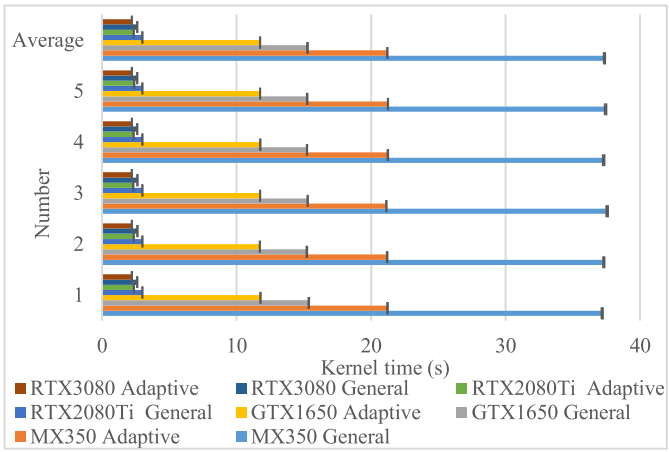


Fig. 3. Comparison of time after adaptive optimization of different GPUs.

Owing to the restriction imposed by the number of registers used in the kernel function, the upper limit of a thread block size is determined by the following formula:

$$\text{Maxnum}_{thread\_block} = \text{ceiling}\left(\frac{Num_{reg\_SM}}{Num_{reg\_thread}}, Num_{thread\_warp}\right) \quad (3)$$

where Num_{reg_SM} is the amount of registers in a GPU SM and Num_{reg_thread} is the amount of registers used in a thread. And, we assume that the thread block size is $Num_{thread\_block} = 64, 128 \ldots, Maxnum_{thread\_block}$, and the actual thread warp used by each thread block is

$$Num_{warp\_block} = Num_{thread\_block} / Num_{thread\_warp}. \quad (4)$$

The size of SM used by each thread in a kernel function is $\text{Size}_{Smemory\_block}$; the allocation unit size of SM is $\text{Size}_{Smemory\_alloc}$; and the SM size of an SM is $\text{Size}_{Smemory\_SM}$. Therefore, the amount of thread blocks that the streaming multiprocessor can allocate is restricted by threads, registers, and SM

$$\begin{cases} \text{Blocks}_{thread} = \frac{Num_{warp\_SM}}{Num_{warp\_block}} \\ \text{Blocks}_{reg} = \frac{Num_{warp\_regs}}{Num_{warp\_block}} \\ \text{Blocks}_{Smemory} = \frac{\text{Size}_{Smemory\_SM}}{\text{ceiling}(\text{Size}_{Smemory\_block},\ \text{Size}_{Smemory\_alloc})} \end{cases}. \quad (5)$$

The GPU thread warp occupancy can be calculated for this number of threads

$$\begin{aligned} &\text{Occupancy}_n \\ &= \text{MIN}\left(Blocks_{thread}, Blocks_{reg}, Blocks_{Smemory}\right) \\ &\times \frac{Num_{warp\_block}}{Num_{warp\_SM}}. \end{aligned} \quad (6)$$

We can calculate the corresponding thread warp occupancy $\text{Max}(\text{Occupancy}_n)$ by traversing the set number of threads and obtain the minimum threads when the occupancy is at its maximum, and Num_{thread_block} is the thread block number. CUDA sets the thread block and thread grid parameters through <<<>>> when the orthorectification kernel function begins, and the thread block size is determined using the aforementioned calculation. Selecting the two-dimensional thread indices $Block_x$ and $Block_y$ should be as average as possible and calculated using the following formula:

$$\begin{cases} Block_x = 2^m \ < \text{ceiling}\left(\sqrt{Num_{thread\_block}}, 2\right) \\ (m = 1, 2, \ldots, n) \\ Block_y = \frac{Num_{thread\_block}}{Block_x} \end{cases}. \quad (7)$$

The corresponding two-dimensional thread grids is: $Grid_x = \text{ceil}(\frac{Img_{cols}}{Block_x})$ and $Grid_y = \text{ceil}(\frac{Img_{rows}}{BLock_y})$,

TABLE I
ORTHORECTIFICATION BASED ON GPU PARALLEL COMPUTATION

Input: original image, RPC parameters, DEM

Steps:

1 Establishment of the RFM.
2 Calculation of four corners to determine the image range and establishment of the simulative orthoimagery.
3 RPC is put into the constant memory of GPU; the original images, DEMs, and simulative orthoimagery are put into the GPU global memory. Adaptively GPU thread parameters are then calculated.
4 For each pixel in the CUDA Kernel:
4.1 Simulative orthoimagery (x, y) affine transform to (B, L).
4.2 (B, L) affine transform to DEM, and interpolate out H.
4.3 (B, L, H) RFM is used to calculate the original image coordinates (X, Y).
4.4 (X, Y) are used to resample the grey value h.
5 Transmit the correction data to the CPU and generate the orthoimagery.

Output: orthoimagery

where: Img_{rows} and Img_{cols} are the numbers of original image rows and columns, respectively

$$\text{War}\, p_{launch} = \frac{Num_{warp\_SM} * Num_{thread\_warp}}{Num_{thread\_block}}. \quad (8)$$

We can use the boundary qualifier "__launch_bounds_" to specify the thread and thread warp to be used in the compiler [34]. By specifying different parameters to increase the kernel occupation, a suitable kernel can achieve better performance optimization. In this study, the size of the thread block calculated above is $\text{threads}_{\text{perB}}$ as the first parameter, and (8) is used to calculate the thread warp $Warp_{\text{launch}}$ as the second parameter. Simultaneously, the L1 cache is opened in the GPU to further improve efficiency.

3) Orthorectification Based on GPU Parallel Computation: This study uses the indirect correction method and accelerates the processing using GPU parallel computation. The indirect correction method begins from the orthorectification side and corrects each pixel. It is relatively independent and does not compete with the direct correction calculation; thus, it is more suitable for GPU parallel calculation.

The GPU parallel computing algorithm for orthorectification is presented in Table I. First, the RFM model is constructed according to the RPC parameters and the original image coordinates according to (9) [35]

$$\begin{cases} Y = \frac{Num_L(B,L,H)}{Den_L(B,L,H)} \\ X = \frac{Num_S(B,L,H)}{Den_S(B,L,H)} \end{cases}. \quad (9)$$

Second, the average elevation of the DEM is used to iteratively calculate the longitude and latitude (B, L) of the four corners and determine the four solstices of the image. The image range can be determined to establish a simulative orthoimagery. Simultaneously, affine transformation parameters can be established according to the spatial resolution. Subsequently, the RPC is put into the constant memory, and the original image, DEM, and simulative orthoimagery are stored in the global memory of GPU, and adaptively calculate the GPU thread parameters simultaneously. Subsequently, the pixelwise (x, y) correction is conducted from the simulative orthoimagery, which uses GPU acceleration to correct each pixel: the corresponding geographical coordinates (B, L) are calculated by the established affine transformation parameters; the corresponding row and columns in the DEM are transformed using (B, L) and the DEM affine parameters; resampling interpolation is conducted to obtain the elevation value H of the longitude and latitude (B, L); the coordinates (X, Y) of the original image side corresponding are calculated according to (B, L, H) through the positive solution formula of the RFM; the grey value calculated by resampling interpolation on this coordinate is given to the orthoimagery pixel (x, y). Finally, the correction data are transmitted to the CPU, and the orthoimagery is generated after a grey value is assigned to the simulative orthoimagery.

III. RESULTS AND ANALYSIS

A. Data and Hardware Environment

The data used in this study were obtained from ZY-3 satellite images. China ZY-3 satellite is the earliest high-precision civil stereoscopic optical satellite, which acquires three-dimensional images of a linear array [36]. In this study, 112 images of Taihu Basin, China were used as the original images for orthorectification. The pixel space size is 3.5 m of the forward/backward images and the nadir image resolution is 2.1 m [37]. The details of the original images are presented in Table II. In addition, public 30-m DEM data were used to obtain the elevation (http://www.gscloud.cn/sources/accessdata/310?pid=302).

The experiment was conducted using Visual Studio 2017. The CPU was Intel i7-10510U, and the GPU was NVIDIA MX 350 2G with Pascal architecture, which matches the computing power of CUDA of 6.1. The GPU parameters used to test the adaptive thread parameters on the different GPUs are presented in Table III.

B. Result of GPU Acceleration Efficiency of a Single Image

The calculation results of the adaptive GPU thread parameters and image blocks for the ZY-3 nadir and forward/backward images are compared in MX 350. As presented in Table IV, the thread block size is 64 (8×8), the thread grid size of the forward/backward image is 2039 × 2048, and that of the nadir image is 3065 × 3072. The value of $\text{Warp}_{\text{launch}}$ is 32, which is the number of warps actually used by the qualifier. As the forward/backward image data are small, the GPU does not require partitioning, and the number of adaptive blocks of the

TABLE II
DETAILED DESCRIPTION OF ZY-3 SATELLITE IMAGES

Image	Gray level resolution (bits)	Spatial resolution (m)	Swath (km)	Size per scene (pixels × pixels)	Volume (GB)
ZY-3 forward/ backward	10	3.5	52.3	16 306 × 16 384	48.38
ZY-3 nadir	10	2.1	21.1	24 516 × 24 576	33.6

TABLE III
GPU PARAMETERS USED IN THE EXPERIMENT

Attribute name	MX 350 2G	GTX 1650 4G	GTX 2080Ti 11G	RTX 3080 10G
SPs	640 (5 SMs×128 cores)	896 (14 SMs×64 cores)	4352 (68 SMs×64 cores)	8704 (68 SMs×128 cores)
SM per SM	96 KB	64 KB	64 KB	100 KB
Max SM per Block	48 KB	64 KB	64 KB	99 KB
L1 cache	48 KB	64 KB	64 KB	128 KB
L2 cache	512 KB	1024 KB	1024 KB	5120 KB
Registers per SM	65 536	65 536	65 536	65 536
Max threads per block	1024	1024	1024	1024
Max threads per SM	2048	1024	1024	1536
Max Warps per SM	64	32	32	48

TABLE IV
CALCULATION RESULTS OF ADAPTIVE GPU THREAD PARAMETERS AND IMAGE BLOCKS

Image	**Adaptive GPU** Size per scene (pixels × pixels)	Image blocks	Thread block size
ZY-3 forward/ backward	16 306 × 16 384	1	64 (8 × 8)
ZY-3 nadir	24 516 × 24 576	2	64 (8 × 8)

TABLE V
CORRECTION TIME COMPARISON

Image Data	CPU run time (s)	General GPU run time (s)	Speedup ratio	Adaptive GPU run time (s)	Speedup ratio
ZY-3 forward/ backward	326.08	16.21	20.12	9.43	34.58
ZY-3 nadir	729.83	37.36	19.56	21.21	34.41

nadir image is two, which is consistent with the number of artificial blocks. Subsequently, we compared the performance of the general and adaptive GPU calculations, and the time recorded in this section is focused on the pixelwise correction, and does not include other overheads, such as reading and writing images and preprocessing. The nvprof performance analysis tool provided by NVIDIA is used to record the correction time when using CUDA for GPU acceleration.

Based on the proposed calculation results of adaptive GPU thread parameters and image blocks, we used bicubic interpolation in the resampling method to conduct a time comparison experiment of orthorectification. The clock () function is used in the CPU to record the quadrupole coordinates require and pixelwise correction time. The calculation of the quadrupole coordinates requires 0.02 s, which only accounts for a very small part of the total running time; this article mainly records the pixelwise corrections, as presented in Table V. For the forward/backward images, the pixelwise correction time varies greatly, the time of pixelwise correction of the CPU single thread is 326.08 s, and the correction time of the general GPU is 16.21 s. The correction time of the proposed adaptive GPU thread parameter method is only 9.43 s, which is 53.17% higher than that of the general GPU, and is 34.58 times faster than that of CPU correction. For the nadir image with more data, the time of CPU correction is 729.83 s and that of general GPU correction is 37.36 s. Adaptive GPU calculation has an acceleration ratio of 34.41 for the CPU, which is 43.22% higher than that of the general GPU. The comparison results demonstrate the adaptive GPU calculation is preferable to those of the CPU and general GPU methods. The blocks set manually are more suitable for the GPU than a conventionally set thread block size. The qualifier parameters calculated according to the determined thread blocks can effectively improve the correction efficiency of the GPU.

In addition, in order to comprehensively evaluate the performance of the proposed method, five rounds independent experiments are conducted and the average time is calculated, as shown in Fig. 3 and Table VI. As presented in the table, as the GPU performance increases, the time required to correct the images significantly decreases. The kernel running time of the conventional GPU implemented on MX 350 before adaptive computing optimization is 37.36 s. Additionally, the running times on GPUs with better performance, such as GTX 1650 and RTX 3080, can be reduced to 15.27 and 2.62 s, thereby increasing the efficiency by 59.13% and 92.99%, respectively. After the

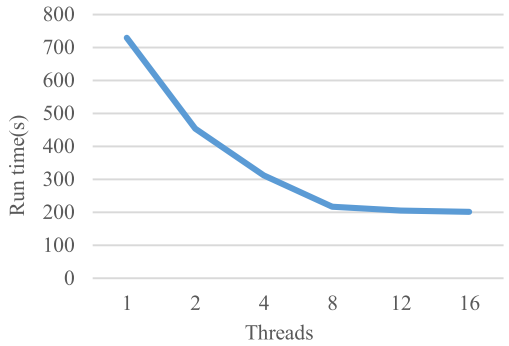


Fig. 4. Correction time of ZY-3 nadir image CPU in multithread.

TABLE VI
ADAPTIVE TEST OF THE ZY-3 NADIR IMAGE IN DIFFERENT GPUS

	MX 350 2G	GTX 1650 4G	GTX 2080Ti 11G	RTX 3080 10G
SM number	5	14	68	68
Image blocks	2	1	1	1
Thread block size	64	64	64	64
$Warp_{launch}$	32	16	16	24
General kernel time	37.36	15.27	2.99	2.62
Adaptive kernel time	21.21	11.75	2.35	2.21
Acceleration ratio	43.22%	23.03%	21.21%	15.72%

optimization of adaptive computing, the kernel function running time on MX 350 was 21.21 s, which was 43.22% higher than that before optimization. After optimization, the correction time on GTX 1650 was 11.75 s, showing an increase of 23.03%; the correction time on GTX 2080Ti was 2.35 s; and that on RTX 3080 was 2.21 s, showing an increase of only 15.72%. Although the improvement reduced after optimizing the GPU with better performance, a better acceleration effect was still obtained.

C. Result of the GPU Acceleration Efficiency of Large-Batch Images

Simultaneously, we individually tested the improvement of batch images correction time and corrected 112 ZY-3 images with the proposed adaptive calculation method. The recording time includes the input, correction, output, and dividing block times of the image. First, the multithreaded correction of a nadir image is tested in the CPU matched with MX 350, as shown in Fig. 4. When the number of threads reaches eight CPU cores, the increase in multithreading time gradually decreases and converges. To prevent CPU overload when running many threads, eight threads are selected to process batch images, and the acceleration times of the GPU are compared. Therefore, the experimental results of orthorectification of 112 ZY-3 images are shown in Fig. 5, and the block-based Wallis color correction is used. As presented in Table VII, the eight thread correction time of the CPU is 15 313.67 s, whereas the correction time of the conventional GPU implementation reaches 3038.638 s. Through

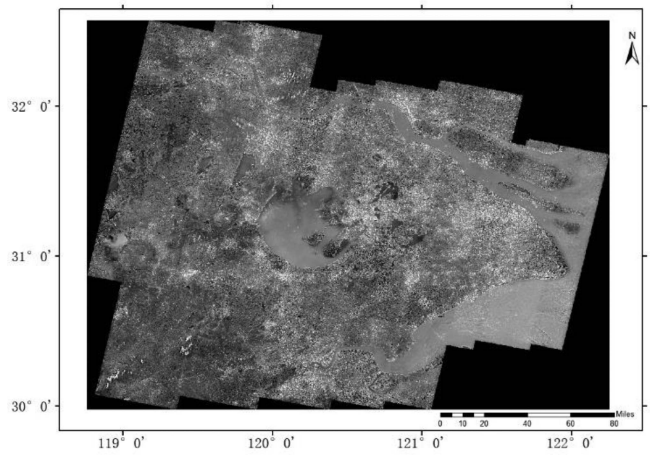


Fig. 5. Experimental results of large-batch orthorectification.

TABLE VII
BATCH CORRECTION TIME OF ZY-3112 IMAGES

	CPU 8 threads	General GPU	Adaptive GPU
Run time (s)	15313.670	3038.638	2084.043

the adaptive calculation in this article, the optimized GPU correction time reached 2084.043 s, which improved the efficiency by 32.6% compared to that of the conventional GPU. Therefore, when mass production of orthoimagery, after optimization using the calculation method of the adaptive GPU thread parameters in this study, the efficiency improvement was better compared to that of the conventional GPU acceleration.

D. Orthorectification Accuracy Based on GPU Parallel Computing

After experimental image preprocessing, we used the bicubic interpolation method to resample the ZY-3 nadir image when the orthorectification was based on GPU parallel computing. The correction accuracy evaluated using the RMSE formula for plane deviation with the same points is presented in Table VIII. We converted the deviation values of longitude and latitude into meters, and the deviation between this correction and the local positioning system (LPS) correction results in the longitudinal and latitudinal directions are 2.2656 and 1.2075 m, respectively, and the overall deviation is 2.5673 m. In this study, no control point was used for orthorectification, and the deviation was approximately one pixel. This effect is good and meets the needs of orthoimagery generation.

IV. DISCUSSION

First, we test the influence of different thread number on the running time of the kernel functions. The maximum threads $Maxthread_{block}$ can be determined by (3), which is 640. Fig. 6 shows that the height of the two-dimensional thread block is 1, and the width is increased by 64 threads each time, from a minimum of 64 to a maximum of 640 correction programs

TABLE VIII
PRECISION RESULTS OF THE SAME POINT

	LPS		Ours		Deviation value	
	latitude	longitude	latitude	longitude	latitude	longitude
1	31.25719106	121.60493878	31.25718311	121.60496308	7.95E-06	-2.43E-05
2	31.25732259	121.60480961	31.25732145	121.60483014	1.14E-06	-2.05E-05
3	31.16209814	121.60694724	31.16210139	121.60696568	-3.25E-06	-1.84E-05
4	31.16198138	121.60579596	31.16198579	121.60582034	-4.41E-06	-2.44E-05
5	31.16264071	121.60665366	31.16264452	121.60667569	-3.81E-06	-2.20E-05
6	31.16180454	121.60593838	31.16180762	121.60596027	-3.08E-06	-2.19E-05
7	31.07677422	121.59689997	31.07678379	121.59691783	-9.57E-06	-1.79E-05
8	30.98553497	121.60660723	30.98555055	121.60661998	-1.56E-05	-1.28E-05
9	30.98385637	121.60445723	30.98387179	121.60447716	-1.54E-05	-1.99E-05
10	30.98385637	121.60445723	30.98387179	121.60447716	-1.54E-05	-1.99E-05
11	31.25626011	121.71027968	31.25625524	121.71030316	4.88E-06	-2.35E-05
12	30.90916343	121.59865604	30.90918680	121.59867116	-2.34E-05	-1.51E-05
13	31.07675197	121.59726725	31.07676093	121.59728744	-8.96E-06	-2.02E-05
14	31.06951384	121.60175592	31.06952287	121.60177789	-9.03E-06	-2.20E-05
15	31.06713296	121.60035367	31.06714510	121.60037614	-1.21E-05	-2.25E-05

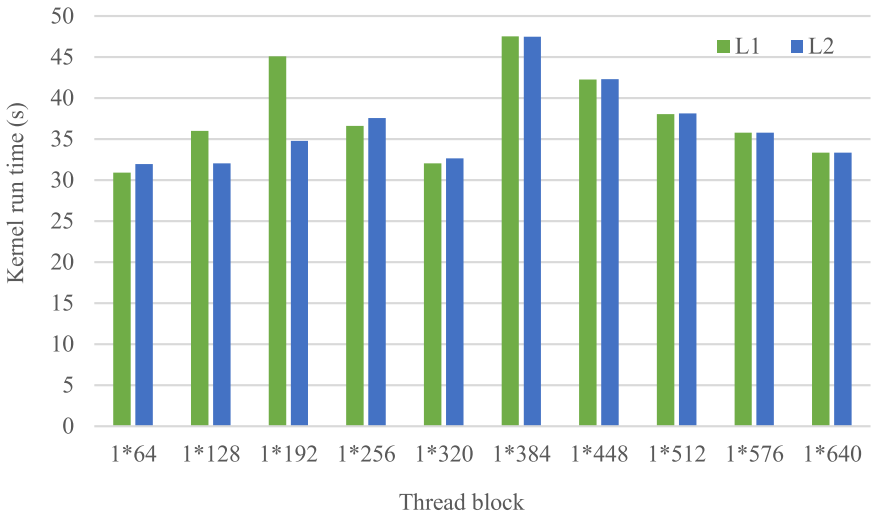


Fig. 6. Kernel function runtime of different thread blocks.

supported by MX 350. The experimental results show that when the line block size is 64, the kernel function runs the least, which is consistent with the results of the adaptive computing thread parameters proposed herein. The increase or decrease in time in Fig. 6 is consistent with the change in threaded warp utilization at values of 64 and 320; in this case, the kernel thread bundle utilization is the highest, thereby enabling the higher performance of the GPU, and the thread parameters are calculated using the thread warp in this study.

The running time of the correction kernel function in the default L2 cache is 31.957 s, whereas that after enabling the L1 cache increased by 3.26% to 30.9152 s, which improved the efficiency. The CUDA kernel function uses a two-dimensional thread index to process images. The number of the two-dimensional thread block is the arithmetic square root of the thread block size or is close to it, with a great effect. The running time results when the calculation thread block size is 64 are shown in Fig. 7. The two-dimensional thread block was 8×8, and the kernel function running time was 31.3118 s with the best performance.

Subsequently, we analyzed the impact of the qualifier on the kernel function time when using the different thread warps. The thread number of the qualifier during conventional implementation was 256, and that determined by the adaptive calculation

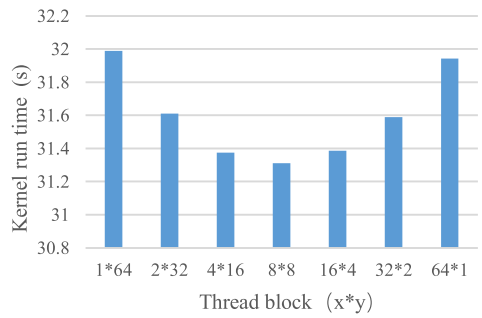


Fig. 7. Two-dimensional threads with a thread block size of 64.

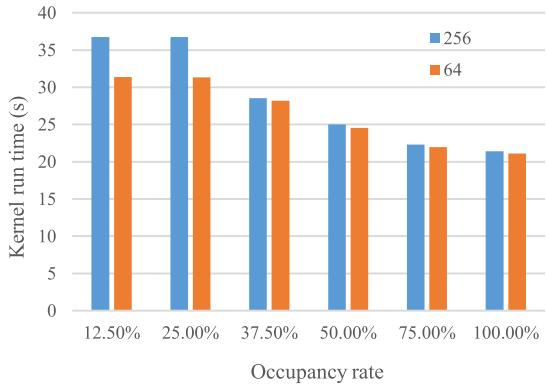


Fig. 8. Kernel function runtime of warp occupancy.

in this study was 64. The second parameter of the qualifier is the number of threaded warps working in the SM, which can change the warp utilization. Fig. 8 shows that as the utilization rate increases, the kernel running time gradually decreases. The highest performance was obtained when the utilization rate was 100%. When implementing the correction method in this study, we observed that the higher the kernel occupancy, better the performance.

V. Conclusion

Orthorectification of images is a time-consuming task. The size of the thread block in most existing parallel computing methods is determined by testing in existing GPU optimization methods, which affects the orthorectification performance on different GPUs. We propose a parallel computing method for orthorectification with adaptive GPU thread parameters, thus solving the problems of adaptively calculating GPU thread parameters in different GPUs and automatically blocking the image to be corrected. After images preprocessing, automatically blocked according to the memory size of the GPU, and the GPU thread parameters are calculated by the physical parameters of the GPU; Finally, orthorectification is achieved based on GPU parallel computing. To verify the efficiency of adaptive computing, experiments were conducted on 112 ZY-3 images, and the results were compared to those using CPU and general GPU methods. The following conclusions were drawn.

1) For a single ZY-3 image, the GPU acceleration by the adaptive calculation method presented in this article is 43.22% higher than that of the general GPU, and 34.41 times faster than the correction time of the CPU. The result of the automatic image blocking was the same as that of the artificial blocking.
2) The method was tested in four different GPUs, and their performances were compared. The results indicate that all the GPUs exhibited a great speed boost. The correction time of the best GPU was only 2.21 s for a single image.
3) For large-batch images, the GPU acceleration by the adaptive calculation method presented in this article is 32.6% higher than that of a general GPU, which provides an adaptive optimization strategy for large-batch image orthorectification.

Notably, the main idea of the proposed method can apply in large batches of image processing tasks; for example, image mosaicing, block adjustment, and dodging uniform colors. In the future, we will attempt to further optimize the utilization of streaming multiprocessors in the GPU to achieve better efficient image processing.

References

[1] G. Zhou, R. Zhang, D. Zhang, J. Huang, and O. Baysal, "Real-time orthorectification for remote-sensing images," *Int. J. Remote Sens.*, vol. 40, no. 5/6, pp. 2451–2465, 2019, doi: 10.1080/01431161.2018.1488296.
[2] T. Wang et al., "Large-scale orthorectification of GF-3 SAR images without ground control points for China's land area," *IEEE Trans. Geosci. Remote Sens.*, vol. 60, Jan. 2022, Art. no. 5221617, doi: 10.1109/TGRS.2022.3142372.
[3] D. Poli and T. Toutin, "Review of developments in geometric modelling for high resolution satellite pushbroom sensors," *Photogrammetric Rec.*, vol. 27, no. 137, pp. 58–73, 2012, doi: 10.1111/j.1477-9730.2011.00665.x.
[4] Y. Zhang, M. Zheng, X. Xiong, and J. Xiong, "Multistrip bundle block adjustment of ZY-3 satellite imagery by rigorous sensor model without ground control point," *IEEE Geosci. Remote Sens. Lett.*, vol. 12, no. 4, pp. 865–869, Apr. 2015, doi: 10.1109/lgrs.2014.2365210.
[5] M. Alkan, G. Buyuksalih, U. G. Sefercik, and K. Jacobsen, "Geometric accuracy and information content of WorldView-1 images," *Opt. Eng.*, vol. 52, no. 2, Feb. 2013, Art. no. 026201, doi: 10.1117/1.oe.52.2.026201.
[6] J. Jeong, C. Yang, and T. Kim, "Geo-positioning accuracy using multiple-satellite images: IKONOS, QuickBird, and KOMPSAT-2 stereo images," *Remote Sens.*, vol. 7, no. 4, pp. 4549–4564, 2015, doi: 10.3390/rs70404549.
[7] J. Cao, B. Yang, and M. Wang, "Jitter compensation of ZiYuan-3 satellite imagery based on object point coincidence," *Int. J. Remote Sens.*, vol. 40, no. 16, pp. 6116–6133, 2019, doi: 10.1080/01431161.2019.1587204.
[8] X. Shen, B. Liu, and Q. Li, "Correcting bias in the rational polynomial coefficients of satellite imagery using thin-plate smoothing splines," *ISPRS J. Photogrammetry Remote Sens.*, vol. 125, pp. 125–131, 2017, doi: 10.1016/j.isprsjprs.2017.01.007.
[9] K.-Y. Oh and H.-S. Jung, "Automated bias-compensation approach for pushbroom sensor modeling using digital elevation model," *IEEE Trans. Geosci. Remote Sens.*, vol. 54, no. 6, pp. 3400–3409, Jun. 2016, doi: 10.1109/tgrs.2016.2517100.
[10] W. Zhao, L. Yan, and Y. Zhang, "An efficient orthorectification method for satellite images," *Front. Comput. Technol. Appl.*, vol. 1, no. 1, pp. 1–8, 2020, doi: 10.33969/twjournals.fcta.2020.010101.
[11] T. Wang et al., "Planar block adjustment and orthorectification of ZY-3 satellite images," *Photogrammetric Eng. Remote Sens.*, vol. 80, no. 6, pp. 559–570, 2014, doi: 10.14358/PERS.80.6.559-570.
[12] D. Qian, "High performance computing: A brief review and prospects," *Nat. Sci. Rev.*, vol. 3, no. 1, p. 16, 2016, doi: 10.1093/nsr/nww009.
[13] P. Balaprakash et al., "Autotuning in high-performance computing applications," *Proc. IEEE*, vol. 106, no. 11, pp. 2068–2083, Nov. 2018, doi: 10.1109/jproc.2018.2841200.
[14] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. benini, "A semisupervised autoencoder-based approach for anomaly detection in high performance computing systems," *Eng. Appl. Artif. Intell.*, vol. 85, pp. 634–644, 2019, doi: 10.1016/j.engappai.2019.07.008.

[15] D. Tychalas and H. Karatza, "High performance system based on cloud and beyond: Jungle computing," *J. Comput. Sci.*, vol. 22, pp. 131–147, 2017, doi: 10.1016/j.jocs.2017.03.027.
[16] D. Esposito, A. G. M. Strollo, E. Napoli, D. De Caro, and N. Petra, "Approximate multipliers based on new approximate compressors," *IEEE Trans. Circuits Syst. I: Regular Papers*, vol. 65, no. 12, pp. 4169–4182, Dec. 2018, doi: 10.1109/tcsi.2018.2839266.
[17] A. Borghesi, M. Molan, M. Milano, and A. Bartolini, "Anomaly detection and anticipation in high performance computing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 4, pp. 739–750, Apr. 2022, doi: 10.1109/tpds.2021.3082802.
[18] E. Christophe, J. Michel, and J. Inglada, "Remote sensing processing: From multicore to GPU," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 4, no. 3, pp. 643–652, Sep. 2011, doi: 10.1109/jstars.2010.2102340.
[19] A. Agathos, J. Li, D. Petcu, and A. Plaza, "Multi-GPU implementation of the minimum volume simplex analysis algorithm for hyperspectral unmixing," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 7, no. 6, pp. 2281–2296, Jun. 2014, doi: 10.1109/JSTARS.2014.2320896.
[20] E. Torti, G. Danese, F. Leporati, and A. Plaza, "A hybrid CPU–GPU real-time hyperspectral unmixing chain," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 9, no. 2, pp. 945–951, Feb. 2016, doi: 10.1109/JSTARS.2015.2485399.
[21] R. Zhang, G. Zhou, G. Zhang, X. Zhou, and J. Huang, "RPC-based orthorectification for satellite images using FPGA," *Sensors*, vol. 18, no. 8, 2018, Art. no. 2511, doi: 10.3390/s18082511.
[22] C. Wu, S. Agarwal, B. Curless, and S. M. Seitz, "Multicore bundle adjustment," in *Proc. IEEE Comput. Vis. Pattern Recognit.*, 2011, pp. 3057–3064, doi: 10.1109/CVPR.2011.5995552.
[23] C.-C. Yeh, Y.-L. Chang, P.-H. Hsu, and C.-H. Hsien, "GPU acceleration of UAV image splicing using oriented fast and rotated brief combined with PCA," in *Proc. IEEE Int. Geosci. Remote Sens. Symp.*, 2018, pp. 5700–5703, doi: 10.1109/IGARSS.2018.8519046.
[24] Y. E. Esin, B. Demirel, Ö. Özdil, and Ş. Öztürk, "Ortho-rectification of hyperspectral camera data with Central Processing Unit and Graphics Processing Unit," in *Proc. 9th Int. Conf. Recent Adv. Space Technol.*, 2019, pp. 465–468, doi: 10.1109/rast.2019.8767856.
[25] C. Dai and J. Yang, "Research on orthorectification of remote sensing images using GPU-CPU cooperative processing," in *Proc. Int. Symp. Image Data Fusion*, 2011, pp. 1–4, doi: 10.1109/isidf.2011.6024247.
[26] Z. Lei, M. Wang, D. Li, and T. L. Lei, "Stream model-based orthorectification in a GPU cluster environment," *IEEE Geosci. Remote Sens. Lett.*, vol. 11, no. 12, pp. 2115–2119, Dec. 2014, doi: 10.1109/lgrs.2014.2320991.
[27] Y. Sun, B. Liu, X. Sun, W. Wan, K. DI, and Z. Liu, "A CPU/GPU collaborative approach to high-speed remote sensing image rectification based on RFM," in *Proc. Remote Sens. Environ.: 18th Nat. Symp. Remote Sens. China*, 2014, vol. 9158, pp. 89–96, doi: 10.1117/12.2063894.
[28] M. Wang, L. Fang, D. Li, and J. Pan, "Using multiple GPUs to accelerate MTF compensation and georectification of high-resolution optical satellite images," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 8, no. 10, pp. 4952–4972, Oct. 2015, doi: 10.1109/jstars.2015.2477460.
[29] L. Penglong, D. Yi, D. Songjiang, L. Ding, J. Ziwei, and X. Yong, "A method of rapid distortion correction for UAV image based on GPU-CPU co-processing technology," in *Proc. IEEE Int. Geosci. Remote Sens. Symp.*, 2018, pp. 5720–5723, doi: 10.1109/igarss.2018.8518556.
[30] H. Kartal, U. Alganci, and E. Sertel, "Automated orthorectification of VHR satellite images by SIFT-based RPC refinement," *ISPRS Int. J. Geo-Inf.*, vol. 7, no. 6, 2018, Art. no. 229, doi: 10.3390/ijgi7060229.
[31] B. Yang, M. Wang, W. Xu, D. Li, J. Gong, and Y. Pi, "Large-scale block adjustment without use of ground control points based on the compensation of geometric calibration for ZY-3 images," *ISPRS J. Photogrammetry Remote Sens.*, vol. 134, pp. 1–14, 2017, doi: 10.1016/j.isprsjprs.2017.10.013.
[32] X. Li et al., "Planar block adjustment for China's land regions with LuoJia1-01 nighttime light imagery," *Remote Sens.*, vol. 11, no. 18, 2019, Art. no. 2097, doi: 10.3390/rs11182097.
[33] X. Zhang, R. Feng, X. Li, H. Shen, and Z. Yuan, "Block adjustment-based radiometric normalization by considering global and local differences," *IEEE Geosci. Remote Sens. Lett.*, vol. 19, Oct. 2020, Art. no. 8002805, doi: 10.1109/lgrs.2020.3031398.
[34] *Cuda C Programming Guide*, Version 10.0, NVIDIA Corp., Santa Clara, CA, USA, 2018.
[35] S. Gholinejad, A. A. Naeini, and A. Amiri-Simkooei, "Optimization of RFM problem using linearly programed ℓ_1-regularization," *IEEE Trans. Geosci. Remote Sens.*, vol. 60, Jan. 2021, Art. no. 5601009, doi: 10.1109/tgrs.2020.3045091.
[36] X. Tong et al., "Optimal selection of virtual control points with planar constraints for large-scale block adjustment of satellite imagery," *Photogrammetric Rec.*, vol. 35, no. 172, pp. 487–508, 2020, doi: 10.1111/phor.12339.
[37] X. Huang et al., "High-resolution urban land-cover mapping and landscape analysis of the 42 major cities in China using ZY-3 satellite images," *Sci. Bull.*, vol. 65, no. 12, pp. 1039–1048, 2020, doi: 10.1016/j.scib.2020.03.003.

Ruyan Zhou received the Ph.D. degree in agricultural bio-environment and energy engineering from Henan Agricultural University, Zhengzhou, China, in 2007.

She is currently an Associate Professor with the College of Information Technology, Shanghai Ocean University, Shanghai, China. Her research interests include photogrammetry and deep learning.

Shangcheng Hu received the B.E. degree in mechanical design, manufacturing, and automation from the Guangling College of Yangzhou University, Yangzhou, China, in 2021. He is currently working toward the M.Eng. degree in computer technology with Shanghai Ocean University, Shanghai, China.

His research interests include remote sensing big data, image processing, and high-performance computing.

Zhonghua Hong (Member, IEEE) received the Ph.D. degree in GIS from Tongji University, Shanghai, China, in 2014.

He has been an Associate Professor with the College of Information Technology, Shanghai Ocean University, Shanghai, China, since 2019. His research interests include satellite/aerial photogrammetry, high-speed videogrammetric, planetary mapping, 3-D emergency mapping, GNSS-R, deep learning, and processing of geospatial big data.

Xiaohua Tong (Senior Member, IEEE) received the Ph.D. degree in geographic information system (GIS) from Tongji University, Shanghai, China, in 1999.

From 2001 to 2003, he was a Postdoctoral Researcher with the State Key Laboratory of Information Engineering in Surveying, Mapping, and Remote Sensing, Wuhan University, Wuhan, China. He was a Research Fellow with The Hong Kong Polytechnic University, Hong Kong, in 2006. From 2008 to 2009, he was a Visiting Scholar with the University of California, Santa Barbara, Santa Barbara, CA, USA. His research interests include trust in spatial data, photogrammetry and remote sensing, and image processing for high-resolution satellite images.

Shijie Liu (Member, IEEE) received the Ph.D. degree in cartography and geographic information engineering from Tongji University, Shanghai, China, in 2012.

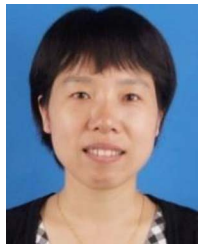
He is currently a Professor with the College of Surveying and Geo-informatics, Tongji University. His research interests include geometric exploitation of high-resolution remote sensing and its applications.

Haiyan Pan received the Ph.D. degree in surveying and mapping from Tongji University, Shanghai, China, in 2020.

She is currently a Lecturer with the College of Information Technology, Shanghai Ocean University, Shanghai, China. Her research interests include multispectral/hyperspectral image classification, multitemporal remote sensing data analysis, and change detection.

Yun Zhang received the Ph.D. degree in applied marine environmental studies from Tokyo University of Maritime Science and Technology, Tokyo, Japan, in 2008.

Since 2011, he has been a Professor with the College of Information and Technology, Shanghai Ocean University, Shanghai, China. His research interests include the study of navigation system reflection signal technique and its maritime application.

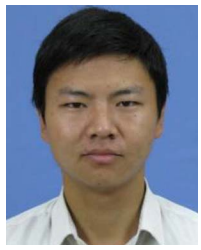


Yanling Han received the B.E. degree in mechanical design and manufacturing and the M.E. degree in mechanical automation from Sichuan University, Sichuan, China, in 1996 and 1999, respectively and the Ph.D. degree in engineering and control theory from Shanghai University, Shanghai, China, in 2005.

She is currently a Professor and is with Shanghai Ocean University, Shanghai, China. Her research interests include the study of ocean remote sensing, flexible system modeling, and deep learning.

Jing Wang received the Ph.D. degree in biomedical engineering from the Department of Biomedical Engineering, Shanghai Jiao Tong University, Shanghai, China, in 2014.

Since 2015, she has been a Lecturer with the College of Information Technology, Shanghai Ocean University, Shanghai, China. Her research interests include computer vision and medical image processing.



Shuhu Yang received the Ph.D. degree in physics from the School of Physics, Nanjing University, Nanjing, China, in 2012.

He has been a Lecturer with the College of Information Technology, Shanghai Ocean University, Shanghai, China, since 2012. His research interests include hyperspectral remote sensing, evolution of the Antarctic ice sheet, and the use of navigational satellite reflections.