Hindawi
Mobile Information Systems
Volume 2022, Article ID 6227894, 11 pages
https://doi.org/10.1155/2022/6227894

Research Article

Implementation of Remote-Sensing Data Processing Platform Based on Computable Storage

Zeyu Qiu,[1] **Jiahong Liu**,[2] **Xu Yang**,[1] **Renpeng Zha**,[1] **Zhen Li**,[1] **and Xishan Bai**[3]

[1]*Information Engineering University, Zhengzhou 450001, China*
[2]*Huazhong University of Science and Technology, Wuhan 430074, China*
[3]*Yunnan Minzu University, Kunming 650504, China*

Correspondence should be addressed to Xishan Bai; 060899@ymu.edu.cn

Received 4 August 2022; Accepted 27 August 2022; Published 9 September 2022

Academic Editor: Imran Khan

Previous researches on accelerating remote sensing data processing are based on traditional von Neumann architecture, which separates storage and computation. Under the architecture, data must be obtained from the storage device first and then transmitted to Field Programmable Gate Array (FPGA) through the system bus. The power consumption caused by the data handling is huge, even exceeding the energy consumption required for data processing. In order to reduce the migration of remote sensing data and alleviate the problems of storage wall and power wall under von Neumann architecture, we design a remote sensing data processing platform based on the system architecture of computable storage, which uses Solid-State Disk (SSD) with computing capability to process the remote sensing data and realize accelerated remote sensing data processing. Based on this platform, applications related to remote sensing data processing such as compression, target detection, and image classification are deployed in SSD to improve the information acquisition rate in remote sensing data. Experimental results show that after compression being offloaded to SSD computing performance is improved by 2.27 times compared with the host CPU. Compared with the host GPU, the target detection speed is improved by 6.25% and the power consumption is reduced by 66.7%. Compared with the host, the detection speed of remote sensing image classification is improved by 78.8%, the power consumption is reduced by 70%, achieving the expected classification effect. The Remote Sensing data Processing Platform based on Computable Storage (CSRSPP) distributes various computing tasks to the SSD for execution, which not only improves the processing speed of computing tasks, but also greatly reduces the power consumption of the platform.

1. Introduction

With the continuous development of remote sensing platforms and the enrichment of remote sensing means, the capacity of remote sensing data is increasing geometrically, reaching TB or even PB level, and the types and formats of data also tend to be diversified. Unstructured and massive data bring a series of problems to storage, management, and application [1]. Under the wave of explosive growth of remote sensing data, how to extract the information in remote sensing data quickly and effectively, reduce the redundancy of information, so as to improve the utilization of information is an urgent problem to be solved. Through various data processing methods such as remote sensing data compression, remote sensing image classification, and remote sensing target detection, the deficiency of insufficient information in a single remote sensing image can be effectively compensated and the quality of remote sensing application products can be improved [2]. Meanwhile, storage devices supporting remote sensing data processing are evolving into high-density, high-bandwidth, and low-delay devices, providing high-quality data guarantee for subsequent tasks such as image classification, target recognition, and change detection. Compared with the traditional disks, SSDDs provide higher read/write performance and storage density, with smaller volume, less static power consumption, and better resistance to physical impact. As a mainstream device for information storage, SSDs are widely

used in disaster monitoring, urban planning, and resource investigation, etc [3, 4].

With the development of storage and data processing technology, computer architecture is constantly updated and evolving under the von Neumann architecture. In this architecture, computing, storage, control, and I/O devices are the building blocks of a computer. Computing and storage are separated, developed independently, and optimized separately. Over the past 20 years, the performance of processors used to perform computing functions has maintained the development rate of Moore's Law by relying on process and multi-core technologies, while the improvement rate of memory performance has only maintained at about 7% per year [5], making the traditional von Neumann architecture face great challenges. On the one hand, the imbalance of performance improvement speed between the current memory and the processor leads to the limited memory bandwidth that cannot guarantee the high-speed data transmission. And the processor is always in the state of waiting for data, which further amplifies the storage wall problem. On the other hand, in high-concurrency computing scenarios such as big data and artificial intelligence, data need to be frequently transported from underlying devices to memory, resulting in serious transmission power consumption [6]. The power consumption generated by data migration is even much greater than that of data processing. Boroumand et al. [7] ran the load of Google applications on user devices and found that an average of 62.7% of the energy was spent on data movement between memory and computing units [7]. When scrolling through Google Doc pages, data movement accounts for up to 77% of the energy consumption.

In order to meet the remote sensing data platform with strict requirements on power consumption and size, and accelerate the data processing of remote sensing applications, FPGAs with outstanding features such as low power consumption, low latency, high performance, small size, and reconfiguration have begun to replace GPUs. It is increasingly used in edge-embedded devices to accelerate the inference process of remote sensing data processing algorithms. González et al. [8] proposed an algorithm based on FPGA for automatically detecting targets to address the challenge of poor instantaneity of target detection in hyperspectral remote sensing images [8]. Shimoda et al. [9] proposed a sparse fully convolutional network based on FPGA for semantic segmentation, and adopted a fully pipelined architecture in hardware implementation [9]. Boskovic et al. [10] implemented a multi-mode hyperspectral image target detection system based on FPGA [10], which could switch between three different target detection algorithms freely. However, these studies are based on the traditional von Neumann architecture, and a large amount of data needs to be transported from the underlying storage device to the memory of the host, which causes the problem of storage wall and power wall.

In order to solve the storage wall and power wall problems of the von Neumann architecture, the concept of transferring computation to memory has gained extensive attention in the academic community and is broadly referred to as near data processing method [11]. The memory includes internal memory and external memory [12]. The internal memory can be accessed in bytes. However, the external memory has a large capacity and cannot be accessed at the granularity of byte addressing. Near data processing seeks to minimize data movement by finding the most appropriate location in the hierarchy for computation [13], taking into account the location of data and information that needs to be extracted from the data. In near data processing, computation can be performed in external memory devices, or in cache, main memory, and persistent storage, as opposed to the traditional way in which data is moved to the CPU and computed within the CPU. The fifth generation (5G) communication has the potential to achieve ubiquitous positioning when integrated with a global navigation satellite system (GNSS). Computable storage is crucial to the construction and development of 5G.

As a system architecture for near data processing, computable storage migrates computation to internal storage devices for execution to reduce data interaction between hosts and peripherals. At present, IBM embeds Blue Gene high-performance computing chips inside storage devices in the data center to create storage nodes with computing capabilities [14]. Samsung launches SmartSSD to deal with workload related to data analysis and storage transactions in storage devices [15]. Eideticom has developed NoLoad, which offloads storage functions such as compression onto the accelerator. Alibaba unloads the database scanning and compression operations into storage devices and applies them to the cloud-native relational database PolarDB [16].

In order to speed up the data processing of remote sensing applications and quickly extract effective information from massive remote sensing data, CSRSPP is designed in this paper, which uses storage devices to accelerate the remote sensing data processing, so as to alleviate the problems of storage wall and power wall in the von Neumann architecture. After the remote sensing data processing is unloaded to the storage device, the application performance is improved and the power consumption is greatly reduced.

2. Design of Remote Sensing Data Processing Platform Based on Computable Storage

CSRSPP includes two parts: the host drive and the SSD, and its overall architecture is shown in Figure 1.

The host is equipped with a block device driver to deliver special computing tasks. SSD consists of multiple independent channels which contribute to high parallelism and rich I/O resources. So it can cope with the problems of large single file, large number of files, and low storage efficiency. Five sub-modules are designed in the SSD to support real-time processing of remote sensing data, including command parser, task scheduler, block device driver, application manager, and acceleration module. There are computing devices inside the SSD, which can process the remote sensing data.

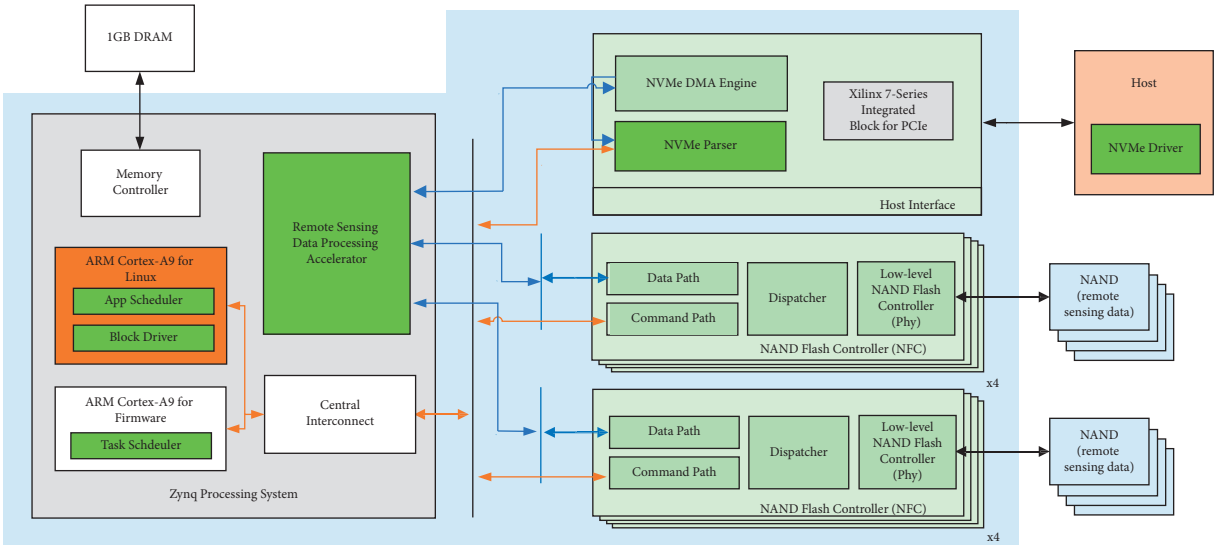

Figure 1: Platform architecture of remote sensing data processing based on computable storage.

3. Operation Principle of SSD

As shown in Figure 2, the SSD consists of host interface logic units, master control chips, Dynamic Random-Access Memory (DRAM) caches, and multiple flash memory chips. The host interface shields the operation on flash memory chips inside the SSD to help the upper-layer storage system seamlessly read and write SSDs. Commonly used interfaces include PCIe, SATA, etc. The main control chip includes processor, memory controller, flash controller, cache manager, providing various methods for processing operations, memory management and organization, and management of flash memory chips. DRAM caches inside the SSD serve to buffer data and accelerate data reading. If the data to be read by the host is in DRAM, there is no need to access the flash memory chips. Instead, the data in DRAM is directly transferred to the host through the Direct Memory Access (DMA) controller. Flash memory chips are used to store data and mounted on different channels. The different channels are connected side by side to the flash controller, enabling parallel access to the SSD.

We use OpenSSD [17] that is an FPGA-based SSD development platform. OpenSSD provides the required IP and basic firmware control logic for the SSD to achieve repeatable programming.

4. Design of Computable Storage System

4.1. Design of Command Parser. The host interacts with the SSD through the NVMe protocol [18]. The NVMe protocol uses multi-queue technology, which can allocate different queues according to task types and scheduling priorities. Each CPU has its own queue to achieve high-performance storage. Each queue of the NVMe protocol is a FIFO pipe that connects the host to the device. The command pipeline sent from the host to the device is called the Submission Queue (SQ), and the command pipeline sent from the device to the host is called the Completion Queue (CQ). The process of an I/O request is that the host first assembles NVMe commands and establishes DMA mappings for transferring data, and then sends the request to the device through SQ. After receiving the request, the device records the corresponding completion result into the I/O completion request, and finally returns the completion result to the host through CQ.

The NVMe protocol uses the doorbell mechanism to inform the SSD controller's CQ whether there is a new request. Each NVMe queue has a doorbell pointer. For SQ, this pointer is the queue tail pointer. After the host submits an I/O request to the SQ, it updates the doorbell pointer in the device register space with the value of the SQ tail pointer. At this point, the SSD controller is notified of a new request and needs to move the NVMe command from the SQ to the command parser for further execution.

After receiving the NVMe command, the command parser needs to parse it. The specific format of the NVMe command is shown in Figure 3. Usually the last 4 fields of the protocol package are not used and can be customized by the user. The 10th and 11th fields are often used, as defined as logical block addresses in read/write commands.

The size of the NVMe command is 64 B, and opcodes larger than 0×80 are reserved options for manufacturers. To support computing tasks of remote sensing data processing, add the following commands: (1) Linux for Remote Sensing Data Processing (LRSDP) heartbeat detection, (2) LRSDP off, LRSDP on, and (3) Computing task delivery. Among them, LRSDP heartbeat detection commands are used to obtain the real-time load of LRSDP in the SSD, LRSDP on and off commands are used to control the startup and shutdown of LRSDP, and computing task delivery commands are used to issue the computing tasks that need to be executed in the SSD.

4.2. Design of Task Scheduler. For the smooth operation of LRSDP, the computing tasks delivered by the host cannot be run directly. On the one hand, the computing tasks delivered by the host may become a system bottleneck due to the time-

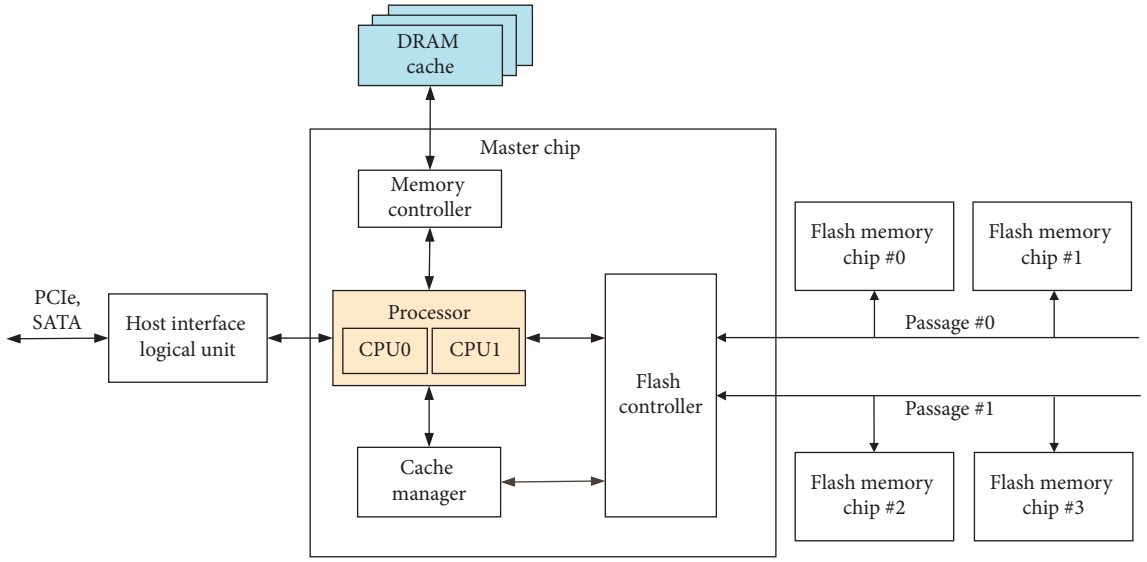

Figure 2: The architecture of SSD.

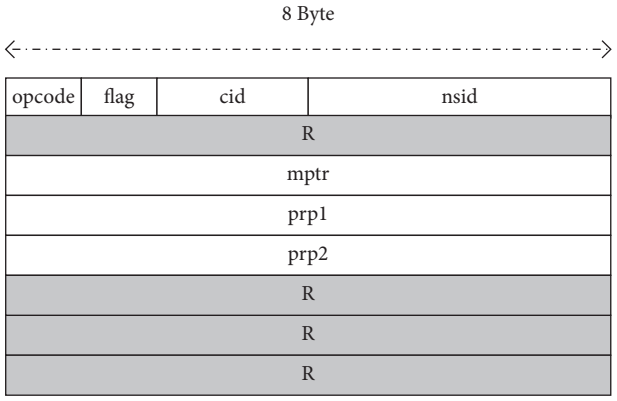

Figure 3: Format of NVMe commands.

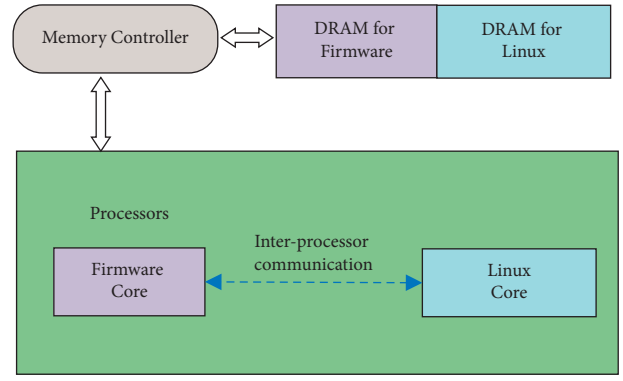

Figure 4: Resource allocation and inter-core communication between firmware core and LRSDP core.

consuming process. On the other hand, the programs executed in the LRSDP may have design flaws, which increase the long-tail latency of the computable storage system. Therefore, the task scheduler is required to schedule the computing tasks delivered by the host. The main job of the task scheduler is to estimate the waiting time of a task. If the expected waiting time is long, the host executes the task or waits for a period of time before delivering the task.

After receiving the request, the task scheduler firstly checks whether the LRSDP has been started. If the LRSDP is not started, the task will be returned to the host. If the LRSDP is started, the depth of the waiting queue of the task and the real-time load of the LRSDP will be evaluated. The real-time load of the LRSDP is obtained through the heartbeat mechanism. The LRSDP periodically informs the task scheduler with the current system information, such as the memory usage and CPU usage. If the queue depth exceeds the threshold or the LRSDP load is heavy, the task is pushed back to the host for processing. Otherwise, the task is inserted into the waiting queue of the task manager, and then sent to the LRSDP for processing.

4.3. Design of Block Device Driver. As shown in Figure 4, it is assumed that there are four processing cores inside the SSD, two of which are used to run firmware and the other two for running LRSDP. The core running the firmware and the core running the LRSDP share the DRAM memory inside the SSD, and they have a different range of physical memory addresses available. Assuming that the low address space is allocated to firmware and the high address space is allocated to LRSDP. The LRSDP block device driver needs to interact with the Flash Translation Layer (FTL) of the firmware through inter-core communication to read and write flash memory information. Therefore, the core of the LRSDP block device driver lies in the inter-core communication.

In a multi-core system, the interrupt controller allows the hardware thread of one CPU to interrupt the hardware thread of other CPUs. This method is called Inter-Processor Interrupt (IPI). The implementation of IPI is based on multi-CPU memory sharing. Using IPI can reduce CPU overload and improve system efficiency effectively. In addition, CPUs in the SSD are generally ARM cores, and the Generic Interrupt Controller (GIC) controller supporting IPI is usually

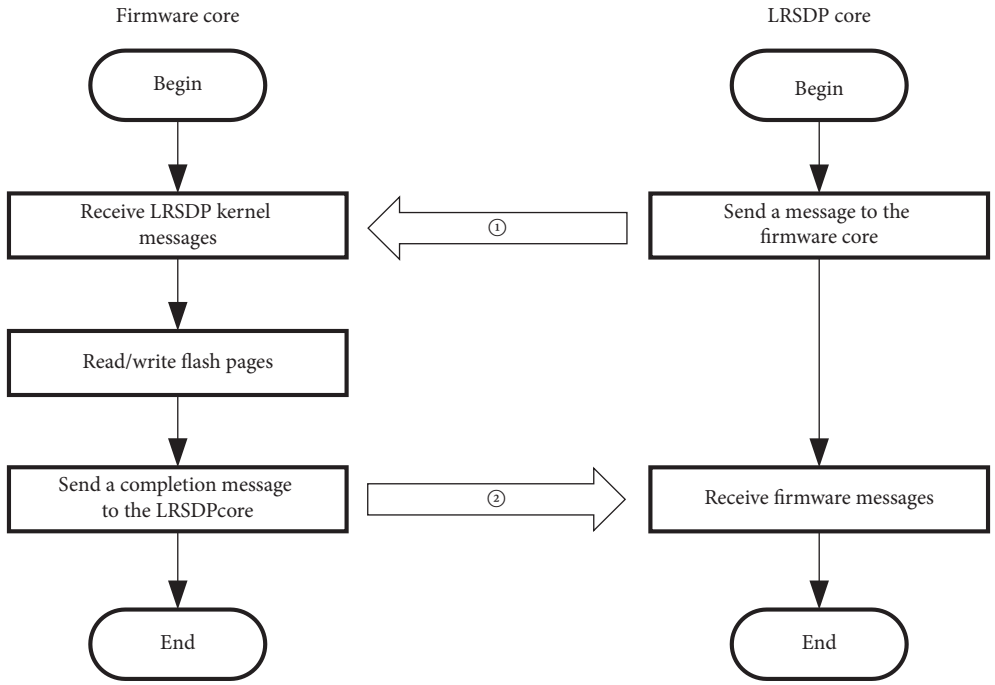

Figure 5: Inter-core communication flow between firmware core and LRSDP core.

configured in the ARM core [19]. Therefore, the IPI between the LRSDP core and the firmware core can be used to communicate and complete data interaction in the computational storage system.

The communication process between firmware core and LRSDP core is shown in Figure 5. The LRSDP core sends packet A to the firmware core, and usually the message length transmitted by the data register is limited. So the valid content sent by LRSDP contains a magic number and the physical address of the actual packet B. The IPI of the firmware core is then triggered to check the validity of packet A's contents. First determine whether the bytes of the packet header match the magic number, if so, continue to read the contents of the packet B corresponding to the physical address. The header of package B contains the opcode and the array length N, followed by the array's content—— [[LBA1, physical address $x1$, sector number $y1$], [LBA2, physical address $x2$, sector number $y2$], . . ., [LBAN, physical address xn, sector number yN]]. Each element of the array consists of the logical block address, the physical address of data read/write, and the number of read/write sectors. The firmware core submits these requests to the FTL I/O queue one by one, and sends an IPI to LRSDP after all I/O requests are completed at the FTL. Finally, the LRSDP core receives the IPI and notifies the file system or application that the I/O is completed.

4.4. Design of Application Manager. The application manager is responsible for the management of LRSDP applications, including the execution of life cycle processes such as saving, starting and stopping applications, and reasonable resource allocation of processes or threads corresponding to computing tasks to prevent a process or thread from blocking the entire system [20].

4.4.1. Application Registration/Saving. The host can register applications with LRSDP, such as issuing script files or executable files. First, the application manager needs to perform trusted code detection on the delivered files to determine whether there is any code that maliciously attacks the operating system or file data. Then, the application manager checks whether the task type delivered by the host already exists in the current system. Finally, the application manager checks whether some parameters given are valid. If all of the above conditions are met, the application manager records the application name and its invocation rules, and saves the application to the root file system of LRSDP for later invocation.

4.4.2. Application Startup and Shutdown. The application starts when a computation task is received from the task scheduler module. First, the application manager creates a process or thread of the corresponding task, and the process/thread enters the running state. After calculating the task result, the process/thread notifies the final processing result to the application manager. The processing results can be recorded in shared memory or files. After the application manager obtains the results, the application is closed and the firmware is notified that the task is completed.

4.4.3. Application Resource Allocation. Although the task scheduler has some control over the computing tasks, the tasks with long waiting time can be directly returned to the host. However, after LRSDP is enabled, measures should be taken when multiple applications preempt the computing resources, memory resources, and I/O resources of the computable storage system. In addition, an application may occupy most of the memory of the computable storage

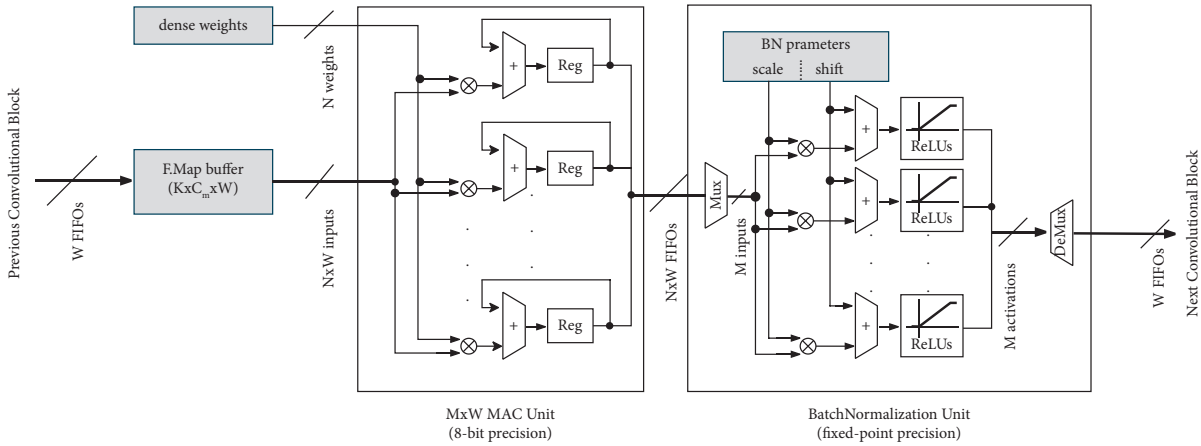

Figure 6: Principle of convolution circuit.

system or occupy the CPU or I/O channel for a long time. Therefore, it is necessary to control the resources occupied by the application.

The host can specify the priority of certain tasks explicitly or adjust the priority of the LRSDP process during the real-time running. In addition, it is also possible to avoid disorderly and unfair resource grabs for the computational storage system by explicitly setting the maximum memory and I/O resources available to a certain application. If there is an abnormal application state, the application manager should end it.

4.5. Design of Application Manager. In the computable storage system, the host can specify remote sensing data processing tasks for acceleration. After the application is started, requests for acceleration tasks are submitted to the acceleration module. The acceleration module completes convolution calculation of all layers in the network model for remote sensing data processing, and its working principle is shown in Figure 6. According to the principle of convolution operation, a feature map vector is convolved with multiple convolution kernel vectors, and the output feature map vector is obtained through the pooling layer [21]. The data input of multiple parallel computing core modules adopts a daisy-chain method, and the feature map vector and convolution kernel vector flow through multiple computing core modules in turn. Therefore, the acceleration module opens up double buffer for this purpose. When the serial number of the convolution kernel vector matches the serial number of the current computing core module, the computing core module caches the vector accordingly, and finally the nth computing core module stores all the weights of the nth convolution kernel. It is used for the calculation of the current batch, and the weight of the nth convolution kernel of the next batch is pre-read, thereby improving the computing performance of the accelerator.

The structure of Convolutional Neural Network (CNN) is getting larger and larger. Due to the resource limitation of FPGA devices, complex CNN models such as Fast R-CNN [22] and ResNet [23], cannot be fully unrolled on hardware, and sometimes even a single convolutional layer cannot be expanded. To solve this problem, the main approach is to map a limited number of processing elements on the FPGA. These processing elements are reused by temporarily iterating over the data. Optimization strategies such as data parallelism, memory access optimization, computational communication overlap, and pipeline optimization are used to maximize the performance of target detection or image classification performed on OpenSSD hardware. In hardware implementation, we use CNN with a simple structure.

5. Experimental Setup

5.1. Experimental Platform. We use the Cosmos plus OpenSSD platform which consists of an XC7Z045 FPGA chip, an Ethernet interface, 1 GB DRAM, an 8-way NAND flash interface, and a PCIe Gen2 8-lane interface. The physical map of the hardware platform is shown in Figure 7.

5.2. Experimental Parameter Settings. The basic parameters of the OpenSSD used in the experiment are shown in Table 1. The flash page size is 16 KB and the logical volume is 1 TB. The read latency of the flash page is 70 μs and the write read latency is 200 μs.

5.3. Experimental Process. Firstly, we create a partition in OpenSSD, and then format it with ext4 file system. Finally, we mount the partition in the host and copy remote sensing data into OpenSSD's NAND flash.

After the remote sensing data is saved to the flash memory, the host initializes the development board with bitstream. After the bitstream initializes the acceleration module on the development board, the host sends a special NVMe command to start LRSDP. After LRSDP is started, the host can continue to issue different types of computing tasks to LRSDP through custom NVMe commands.

When LRSDP receives a computing task, it will continuously transfer data to Remote Sensing Data Processing Accelerator (RSDPA). RSDPA calls the corresponding hardware resources for calculation according to the specific type of computing task. After calculation, the data is stored

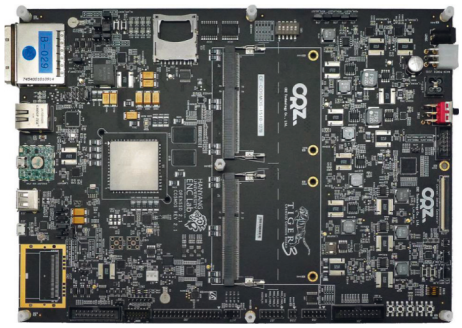

Figure 7: OpenSSD hardware platform.

Table 1: Detailed parameters of SSD.

Parameters	Configuration
Flash page size	16 KB
Page number in flash block	256
Number of flash channels	8
Logical volume	1 TB
Read latency of flash page	70 μs
Write latency of flash page	200 μs
Erasure delay of flash block	2 ms

in Block Random Access Memory (BRAM) of the FPGA, and LRSDP can copy and read the execution results in BRAM to DRAM of OpenSSD. An end signal is sent to notify the host that the computing task is completed. After receiving the end signal, the host can actively read the processing result of the task through the PCIe interface, and read the final data processing through the DMA controller.

6. Results and Discussion

This section comprehensively evaluates the feasibility of CSRSPP proposed in this paper through experiments. Based on different large-scale remote sensing datasets, we design remote sensing data processing tasks by computational storage, and observe the efficiency of the platform in remote sensing data compression, target detection, and image classification. The accuracy rate and the resource utilization rate of the platform are tested to verify the effectiveness of CSRSPP, and compared with the power consumption under the traditional von Neumann architecture.

7. Data Compression

This section explores the performance gains of compressed applications using computable storage technology at different in-disk bandwidths, i.e., testing 3 different channel counts (2, 4, and 8) [24]. The remote sensing images based on BMP format were converted into jpeg format images to reduce the storage overhead of remote sensing data.

The size of the compression experiment test set is about 20.5 GB, all of which are images in BMP format. The experiment was based on the assumption of low time locality of data. Data needed to be read from the flash memory in the SSD each time, and the compressed result was directly stored on the flash memory. The execution time of compression application consists of I/O time and computation time. The shorter the application execution time, the better the compression performance. Table 2 describes the performance of data compression applications at different channel numbers.

Table 2: The execution time of compression application at different channels.

Platform	Channel number	I/O time (s)	Computation time (s)
HOST	2	36.75	204.25
	4	21.63	209
	8	12.2136	200.9643
CSRSPP	2	36.12	60.8
	4	21.21	62.7
	8	11.7936	64.6

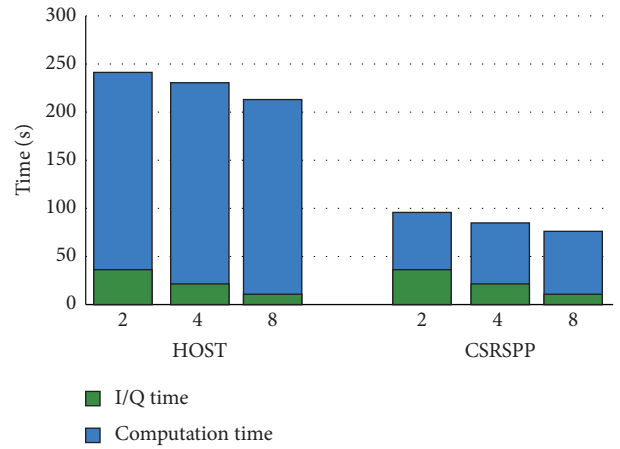

Figure 8: Data compression performance under different channels.

The results of the performance comparison between the host and the CSRSPP performing data compression tasks under different channels are shown in Figure 8.

Combined with the test results in Table 2 and Figure 8, it can be seen that under different channels, the I/O rate of CSRSPP in-disk image reading is 1.7% ~ 3.5% faster than that of off-disk image reading on the host, which makes in-disk I/O process more streamlined than off-disk. The computation speed of CSRSPP in-disk image compression is 2.11 ~ 2.35 times faster than that of off-disk reading on the host. This is because the parallel computing resources of FPGA are utilized, and the pixels of multiple different blocks can be rapidly compressed in parallel.

8. Target Detection

We used the hardware resources of OpenSSD to process CNN in parallel to finish object detection in remote sensing images. The GPU also used the same neural network for object detection.

8.1. Visualization of Detection Results. In the experiment, the dataset for object detection in aerial images DOTA-1.0 published by Wuhan University [25] and the high resolution ship dataset HRSC2016 published by Northwestern

(a) (b) (c) (d) (e) (f)

Figure 9: Target detection visualization results based on DOTA-1.0 and HRSC2016 datasets.

Polytechnical University were used [26]. DOTA-1.0 contains a total of 2806 aerial images containing objects of various scales, orientations, and shapes in 15 common categories for training and testing. HRSC2016 has a total of 1079 images, and the only marked target is the ship, which is used for the test of platform detection performance.

The visualization results are shown in Figure 9, where Figures 9(a)–9(c) are generated by DOTA-1.0 dataset, and Figures 9(d)–9(f) are generated by HRSC2016 dataset. It can be seen that the target detection performance of the platform performs well on different datasets.

8.2. Detection Efficiency Test. In this experiment, we tested the target detection effect under the DOTA-1.0 and HRSC2016 datasets to verify the effectiveness and universality of the platform for detection tasks. The mean Average Precision (mAP) and Frames Per Second (FPS) are selected as objective evaluation indicators. The detection results are shown in Table 3.

It can be seen from the objective index results in Table 3 that the accuracy and detection efficiency of the two datasets are both good when CSRSPP accelerators are used to perform target detection, indicating that the platform is effective and universal for target detection tasks. The mAP of DOTA-1.0 dataset is smaller than that of HRSC2016 dataset, because DOTA-1.0 dataset has more image types than HRSC2016 dataset.

8.3. Comparison on Detection Speed. As a comparative experiment, the GPU system used 16 bit floating-point numerical precision on the multiply-accumulate setting and 32 bit floating-point numerical precision on RSDPA. The power measurement included the CPU and other peripherals in the FPGA and GPU, which were measured under load. For comparison, we used full-resolution images for the GPU system as well as the FPGA system.

As shown in Table 4, our FPGA system achieves 34FPS on images with 500×500 resolution, especially with only a 0.4% loss in accuracy and a power consumption of only 20 W. The GTX 1080Ti dedicated accelerator achieves 32 FPS under the condition of 60 W power consumption. Compared with the full-resolution system on GPU, the power consumption of our system is 66.7% lower. For the detection speed of images with 500×500 resolution, the power consumption performance is improved by 2.21 times. Meanwhile, the architecture of computable storage also has better performance by using RSDPA compared with GPU processing. In general, the remote sensing data processing platform based on OpenSSD greatly reduces the power consumption of the entire system on the basis of a slight reduction in speed.

Table 3: Target detection effects under different datasets.

Dataset	mAP (%)	FPS
DOTA-1.0	74.62	56.7
HRSC2016	83.17	52.3

Table 4: Comparison between two platforms on target detection.

Platform	CPU + OpenSSD	CPU + GPU (GTX 1080Ti)
Freq.[H2]	100	1481
Frame resolution	500×500	500×500
Tile size	250×250	250×250
Precision (MACs)	INT8	FP16
Precision (BNs)	Fixed (32 and 16 bit)	FP32
Precision (%)	97.9	98.3
Speed (FPS)	34	32
Power (W)	20	60
Efficiency (FPS/W)	1.70	0.53

9. Image Classification

In order to verify the information extraction and acceleration effect of CSRSPP on remote sensing data information, CNN was also used to realize remote sensing image classification.

9.1. Visualization of Classification Results. The remote sensing data adopted was from the downloaded Landsat-8. The dataset containing 800 remote sensing images was obtained by cropping and sorting, with a total amount of 22.9 GB, and all images were in JPG format. 640 images were used for training and 160 images were used for testing. The visual result of image classification is shown in Figure 10.

Based on the observation of the visualized classification results of remote sensing image classification in Zhengzhou, CSRSPP achieves a good classification effect for remote sensing images on the five basic categories of mountain, construction land, water Body, green land, and bare land.

9.2. Classification Efficiency Test. In order to test the acceleration effect of the platform on remote sensing classification, 16 bit floating-point numerical accuracy was used on the multiplication-accumulation settings of the GPU system and 32 bit floating-point numerical accuracy was used on RSDPA. The power measurement included CPU and other peripheral devices in FPGA and GPU, which were measured under load. Overall Accuracy (OA) and Kappa coefficient were tested. The test results are shown in Table 5.

Table 5 shows that OA and Kappa are slightly lower than GPU due to limited floating-point accuracy calculation in image classification under the OpenSSD platform. However, the classification speed of RSDPA is 78.8% higher than that of GPU, and the system power consumption of OpenSSD is 70.0% lower than that of GPU. The amount of data returned by the OpenSSD platform based on computable storage is also greatly reduced. In the traditional von Neumann architecture, 4.58 GB of data needs to be returned to the memory. However, the OpenSSD platform based on computable storage only needs to transfer 0.77 GB of data, which

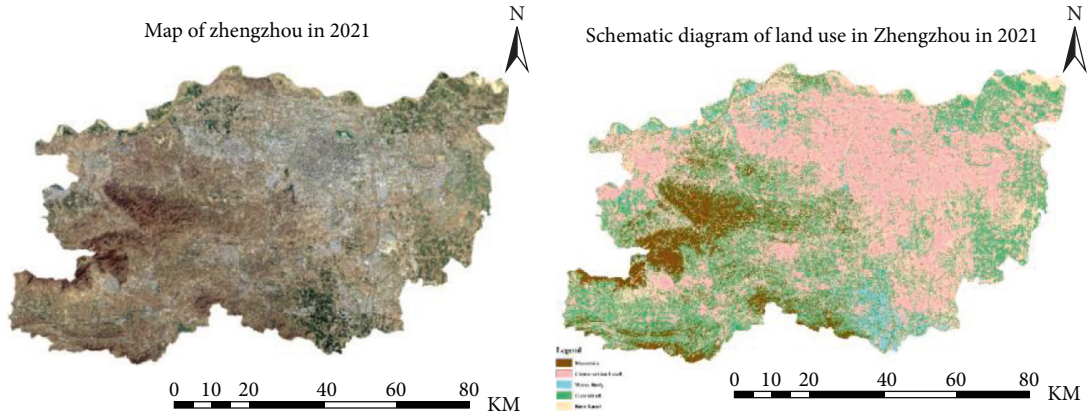

Figure 10: Image classification visualization results based on Landsat-8 dataset.

Table 5: Objective indicator results of image classification.

Index	OpenSSD	GPU
Classification time (s)	468	837
Power	18 W	60 W
OA (%)	84.52	86.44
Kappa	0.81	0.83

Table 6: Comparison on resource utilization of different modules.

Module	#	LUT	FF	BRAM	DSP
Flash controller	8	11031	7539	21	0
NVMe interface	1	8586	11455	28	0
RSDPA	1	76344	19144	157	235
Total	1	212099	152908	384	235

decreases by 83.2% and further reduces the power consumption of CSRSPP.

10. Resource Utilization of the Platform

The placement and routing were completed with Vivado 2018.3. Table 6 shows the hardware utilization of CSRSPP. It reports the resources overhead of the flash controller, NVMe controller, and the RSDPA accelerator module.

However, CSRSPP is more energy-efficient (QPS/Watt) than a GPU-integrated system by 6.56 times. More importantly, the CSRSPP is implemented with FPGA and the operating frequency is only 100 MHz, which can greatly facilitate the construction of 5G. The performance will be further improved if the CSRSPP is implemented with escalated operating frequency or ASIC.

11. Conclusions

In this paper, we design a remote sensing data processing platform based on computable storage, which can proactively perform various remote sensing image processing tasks with low delay, low power consumption, and high precision inside SSD. Based on the OpenSSD platform, we design and implement remote sensing data compression, remote sensing image classification, remote sensing target detection, and other applications inside SSD. Our prototype shows that in the practice of remote sensing data compression, the calculation speed of image compression in CSRSPP is 2.11 ~ 2.35 times faster than that of off-disk reading on host. In the application of target detection, CSRSPP reduces latency by an average of 6.25% and saves power consumption by 66.7% compared to GPU. For remote sensing image classification applications, the detection accuracy under CSRSPP decreases slightly, but the average delay is reduced by 78.8%. CSRSPP distributes various computing tasks to the SSD for execution, which not only improves the speed of information extraction on remote sensing data, but also greatly reduces the power consumption of data migration.

Data Availability

The data that support the findings of this study are available from DOTA-1.0 dataset, HRSC2016 dataset, and Landsat-8 dataset, which are publicly available.

Ethical Approval

The authors declare that they have no human participants, their data, or biological material used in this work.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Authors' Contributions

Z. Q. and J. L. conceptualized the study; Z. Q. and R. Z. developed the methodology; Z. Q. helped with software; Z. Q., X. Y., and R. Z. validated the study; J. L. carried out formal analysis; Z. Q. investigated the study; Z. Q. and R. Z. collected the resources; Z. L. curated the data; Z. Q. wrote and prepared the original draft; X. Y. wrote, reviewed, and edited the study; X. B. visualized the study; R. Z. supervised

the study; X. B. administrated the project; R. Z. carried out funding acquisition. All authors have read and agreed to the published version of the manuscript.

Acknowledgments

This research was funded by the National Natural Science Foundation of China, Grant no. 62171470.

References

[1] M. E. Hereher and H. Ismael, "The application of remote sensing data to diagnose soil degradation in the Dakhla depression–Western Desert, Egypt," *Geocarto International*, vol. 31, no. 5, pp. 527–543, 2016.

[2] M. E. Huq, A. Dughairi, and M. M. Rahman, "Remote sensing big data: challenges, opportunities, management and application introduction," *J Arab Human Sci*, vol. 13, pp. 1–15, 2020.

[3] Z. Yang, J. Bhimani, J. Wang, D. T. Evans, N. Mi, and Z. Yang, "Automatic and scalable data replication manager in distributed computation and storage infrastructure of cyber-physical systems," *Scalable Computing: Practice and Experience*, vol. 18, no. 4, pp. 291–312, 2017.

[4] S. H. Kim, S. J. Lee, S. H. Kim, S. E. Kang, D. S. Lee, and S. R. Lim, "Environmental effects of the technology transformation from hard-disk to solid-state drives from resource depletion and toxicity management perspectives," *Integrated Environmental Assessment and Management*, vol. 15, no. 2, pp. 292–298, 2019.

[5] A. Barbalace, A. Iliopoulos, H. Rauchfuss, and G. Brasche, "It's time to think about an operating system for near data processing architectures," in *Proceedings of the Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, pp. 56–61, Canada, 2017.

[6] M. Gao, G. Ayers, and C. Kozyrakis, "Practical near-data processing for in-memory analytics frameworks," in *Proceedings of the 2015 International Conference on Parallel Architecture and Compilation*, pp. 113–124, PACT, San Francisco, CA, USA, 2015.

[7] A. Boroumand, S. Ghose, Y. Kim et al., "Google workloads for consumer devices: mitigating data movement bottlenecks," in *Proceedings of the Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 316–331, Williamsburg, VA, USA, 2018.

[8] C. González, S. Bernabé, D. Mozos, and A. Plaza, "FPGA implementation of an algorithm for automatically detecting targets in remotely sensed hyperspectral images," *Ieee Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 9, pp. 4334–4343, 2016.

[9] M. Shimoda, Y. Sada, and H. Nakahara, "FPGA-based inter-layer pipelined accelerators for filter-wise weight-balanced sparse fully convolutional networks with overlapped tiling," *Journal of Signal Processing Systems*, vol. 93, no. 5, pp. 499–512, 2021.

[10] Đ. Bošković, M. Orlandić, and T. A. Johansen, "A reconfigurable multi-mode implementation of hyperspectral target detection algorithms," *Microprocessors and Microsystems*, vol. 78, Article ID 103258, 2020.

[11] R. Kaplan, L. Yavits, and R. Ginosar, "From processing-in-memory to processing-in-storage," *Supercomput Front Innov*, vol. 4, pp. 99–116, 2017.

[12] J. Ou, J. Shu, and Y. Lu, "A high performance file system for non-volatile main memory," in *Proceedings of the Proceedings of the Eleventh European Conference on Computer Systems*, pp. 1–16, 2016.

[13] A. Lerner and P. Bonnet, "Not your grandpa's SSD: the era of Co-designed storage devices," in *Proceedings of the Proceedings of the 2021 International Conference on Management of Data*, pp. 2852–2858, Shaanxi, 2021.

[14] R. Haring, M. Ohmacht, T. Fox et al., "The IBM Blue Gene/Q compute chip," *IEEE Micro*, vol. 32, no. 2, pp. 48–60, 2012.

[15] J. H. Lee, H. Zhang, V. Lagrange, P. Krishnamoorthy, X. Zhao, and Y. S. Ki, "SmartSSD: FPGA accelerated near-storage data analytics on SSD," *IEEE Computer Architecture Letters*, vol. 19, no. 2, pp. 110–113, 2020.

[16] W. Cao, Y. Liu, Z. Cheng et al., "{POLARDB} meets computational storage: efficiently support analytical workloads in {Cloud-Native} relational database," in *Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST 20)*, pp. 29–41, Santa Clara, CA, USA, 2020.

[17] J. Kwak, S. Lee, K. Park, J. Jeong, and Y. H. Song, "Cosmos+ openssd: rapid prototype for flash storage systems," *ACM Transactions on Storage*, vol. 16, no. 3, pp. 1–35, 2020.

[18] B. Gu, A. S. Yoon, D.-H. Bae et al., "Biscuit: a framework for near-data processing of big data workloads," *ACM SIGARCH - Computer Architecture News*, vol. 44, no. 3, pp. 153–165, 2016.

[19] K. K. Matam, G. Koo, H. Zha, H.-W. Tseng, and M. Annavaram, "GraphSSD: graph semantics aware SSD," in *Proceedings of the Proceedings of the 46th International Symposium on Computer Architecture*, pp. 116–128, Arizona, Phoenix USA, 2019.

[20] J. Yang, D. B. Minturn, and F. Hady, "When poll is better than interrupt," in *Proceedings of the Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST)*, p. 3, San Jose, CA, USA, 2012.

[21] Y. Yang, D. Zhu, T. Qu, Q. Wang, F. Ren, and C. Cheng, "Single-stream CNN with learnable architecture for multi-source remote sensing data," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–18, 2022.

[22] R. F. R.-C. N. N. Girshick, *Comput Sci*, pp. 1440–1448, 2015.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[24] C. Lee, D. Sim, J. Hwang, and S. Cho, "{F2FS}: a new file system for flash storage," in *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST 15)* pp. 273–286, Santa Clara, CA, USA, 2015.

[25] G. Xia, X. Bai, J. Ding et al., "DOTA: a large-scale dataset for object detection in aerial images," in *Proceedings of the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3974–3983, 2018.

[26] Z. Liu, H. Wang, L. Weng, and Y. Yang, "Ship rotated bounding box space for ship extraction from high-resolution optical satellite images with complex backgrounds," *IEEE Geoscience and Remote Sensing Letters*, vol. 13, no. 8, pp. 1074–1078, 2016.