A Dynamic Acceleration Method for Remote Sensing Image Processing Based on CUDA

Xianyu Zuo[1,3], Zhe Zhang[1,3], Baojun Qiao[1,2], Junfeng Tian[2,3(✉)], Liming Zhou[2,3], and Yunzhou Zhang[4]

[1] Henan Key Laboratory of Big Data Analysis and Processing, Henan University, Kaifeng 475004, People's Republic of China
[2] Henan Engineering Laboratory of Spatial Information Processing, Henan University, Kaifeng 475004, People's Republic of China
[3] College of Computer and Information Engineering, Henan University, Kaifeng 475004, China
[4] National Cultural Heritage Administration, Beijing 100010, People's Republic of China

Abstract. The incredible increase in the volume of remote sensing data has made the concept of Remote Sensing as Big Data reality with recent technological developments. Remote sensing image processing is characterized with features of massive data processing and intensive computation, which makes the processes difficult. To optimize the remote sensing image processing for GPU, compute unified device architecture (CUDA) is widely used to implement remote sensing algorithms. However, the usage of GPU in remote sensing image processing has been constrained by the complexity of its implementation and configuration. Therefore, how to take fully advantage of the parallel organization of GPU architecture is awfully challenging. In this paper, a dynamic adaptive acceleration (DAA) method is proposed to determine calculation parameters of GPU adaptively and preprocess the input remote sensing images on host dynamically. By this method, we determine calculation parameters according to the hardware parameters of GPU firstly. And then, the input remote sensing images are reconstructed based on the calculation parameters. Finally, the preprocessed image blocks are arranged to stream tasks and executed on GPU respectively. Effectiveness of the proposed DAA method in accelerate remote sensing algorithm with point operations were verified by experiments in this paper, and the experimental results indicated that the DAA method can obtain better performance than traditional methods.

Keywords: Remote sensing data · Image processing · CUDA stream · Dynamic acceleration

1 Introduction

With the rapid development of the modern remote sensing technology, remote sensing data having both spectral and spatial information are provided by hyperspectral sensors [1]. And with the increased spatial and spectral resolution of sensors, the amount of remote sensing data has dramatically increased. Remote sensing data has met the basic

H. Song and D. Jiang (Eds.): SIMUtools 2020, LNICST 369, pp. 409–426, 2021.
https://doi.org/10.1007/978-3-030-72792-5_34

characteristics of big data which defined as 3V: volume, velocity, and variety [2, 3]. Thus, remote sensing data reaching high dimensions is defined as remote sensing big data and analyzed in many studies [2, 4, 5]. While acquisition of remote sensing data is no longer the most significant issue, but the processing performance of remote sensing images. Remote sensing image processing is characterized with features of massive data processing, intensive computation, and complex processing algorithms which make the processing of remote sensing image difficult and inefficient [6]. In order to obtain a better performance, GPU (Graphics Processing Unit) is widely used in the field of remote sensing image processing.

GPU are successful accelerators as they show high data throughput with sustainable power budget and is well supported by the SIMT (Single Instructions, Multiple Threads) programming models such as CUDA and OpenCL [6]. In past studies, it has been proved that the time consumption of many remote sensing image processing algorithms can be significantly reduced when optimized for GPU. For example, Wu *et al.* [7] presented a computationally efficient parallel implementation for a spectral-spatial classification method that achieved significant acceleration factors higher than 70 times with NVIDIA GPU. Li *et al.* [8] focused on the most time-consuming part of the manifold learning algorithms designed for HIS data analysis, accelerated by GPU and obtained an excellent speedup performance. Ayomide yusuf *et al.*[7] had surveyed the studies about the usages of GPU in hyperspectral images, and concluded that the implementation of parallel algorithms on GPU has significantly improved the classification of hyperspectral images.

Moreover, CUDA stream have further improved the performance of GPU based programs. Without any synchronization and based on architectural capabilities, NVIDIA's CUDA stream allows some processes to run simultaneously [8]. Leonel Toledo *et al.* [9] illustrated that using dynamic parallelism and CUDA streams were able to achieve up to 30% speedups. HuiChao Hong *et al.* [10] implemented the method based on CUDA streams to compute the GLCM of an image and 50 times faster than ever before. However, in spite of the excellent performance that GPU based remote sensing applications have achieved, how to fully take advantage of CUDA streams in remote sensing processing and reduce the utilization complexity is awfully challenging. In order to take fully advantage of CDUA streams, Mohamad B R *et al.* [8] proposed a method to predict program parts which are able to be overlapped on CUDA streams. However, the parallel processes were carefully designed and optimized with the knowledge of a specific algorithm in most studies, but rarely consider to propose an adaptive and efficient method that can determine execution parameters of CUDA streams according to GPU hardware and features of the input data to be processed dynamically.

Concerning the advantages and shortcomings aforementioned of CUDA streams, this paper proposed a dynamic adaptive acceleration (DAA) method for remote sensing image processing to improve the performance of GPU based programs and reduce the utilization complexity of CUDA streams. The DAA method can determine the appropriate number of streams automatically and determine the suitable size of image blocks adaptively according to the GPU hardware parameters and the remote sensing images to be processed. With this method, developers can take advantages of CUDA streams more conveniently for remote sensing image processing. The main contributions of this paper are as following:

(1) Architecture of CUDA-enabled GPU was introduced, and theoretical analysis about the acceleration mode and execution characteristics of typical GPU parallel models, the multi-thread parallel model and the multi-stream parallel model, were performed respectively. Moreover, the advantage and disadvantage of multi-stream parallel model based on CUDA streams were explained straightly.
(2) According to the correlations between GPU hardware and CUDA programming model, an adaptive strategy which aim to determine the appropriate calculation parameters for CUDA based programs was proposed in this paper.
(3) This paper proposed an DAA method to maximize resource utilization of GPU and is effective to accelerate the remote sensing image processing based on the previous adaptive strategy and the characteristics of remote sensing images. The theoretical analysis and experimental results have verified the superiority of the DAA method compared to general methods.

The remainder of this paper is organized as follows. Section 2 provides the related works about architecture of CUDA-enabled GPU and the typical parallel models of CUDA. Section 3 presents the proposed DAA method for remote sensing image processing on GPU and Sect. 4 validates the method with experiments. Section 5 concludes this paper and describes the future works.

2 Related Work

2.1 Architecture of CUDA-Enabled GPU

Generally, A GPU is composed of massive parallel processors with high floating point performance and memory bandwidth, and has high data throughput [11, 12]. In terms of NVIDA GPUs, SP (Streaming Processor) is the basic unit for the execution of the GPU, and multiple SPs and memories make up the SM. And, multiple SMs, memories, and interconnection networks make up the whole GPU, as shown in Fig. 1. Moreover, GPU is specialized for highly parallel computation, which is exactly what image processing is about, and therefore designed such that more processors are devoted to data processing rather than data caching and flow control [13]. Furthermore, in order to apply GPU to the general purpose computing, CUDA was introduced by NVIDIA in November 2006, which is a general purpose parallel computing platform and programming model that leverages the parallel compute engine in NVIDIA GPUs to solve many complex computational problems in a more efficient way [13, 14]. Mapping images or data to be processed to the parallel processors, the burdensome tasks can be accelerated by GPU effectively. To take fully advantage of the parallel organization of GPU and obtain the best speedup, it is suggested to design and optimize the parallel process carefully with the knowledge of GPU hardware architecture and the features of image processing algorithms [6].

2.2 Parallel Models of CUDA

In this section, the two parallel models of CUDA are discussed and analyzed. In order to demonstrate the characteristics of the models clearly, the time consumption of data transfers between host (CPU) and device (GPU) is not considered.

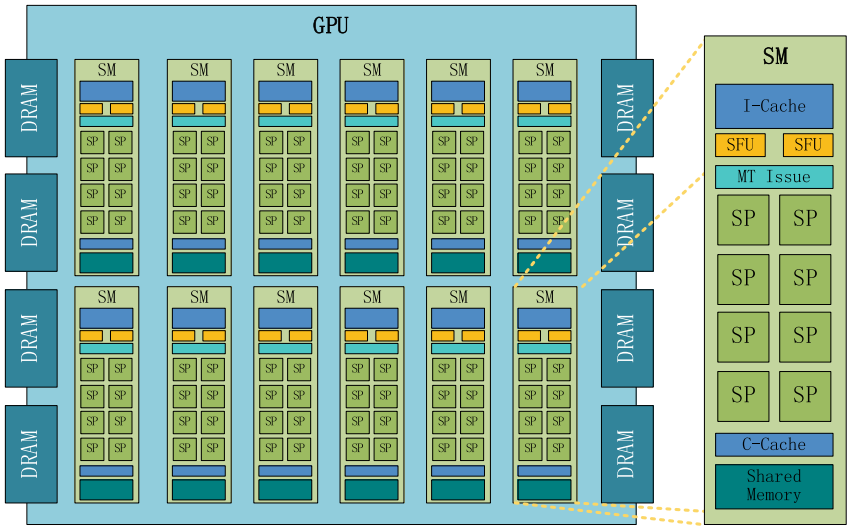

Fig. 1. The general hardware architecture of CUDA-enabled GPU produced by NVIDIA.

2.2.1 Multi-thread Parallel Model

The SIMT architecture of CUDA is akin to SIMD (Single Instruction, Multiple Data) vector organizations in that a single instruction controls multiple processing elements [13]. As described above, the CUDA-enabled GPU generally has a large number of processors compared to CPU. And therefore, the simplest implementation for applications which accelerated by CUDA is to divide tasks into subtasks and then executed by thread blocks. And this method which called multi-thread parallel model, as shown in Fig. 2, is widely used in order to take advantage of the abundant computing cores.

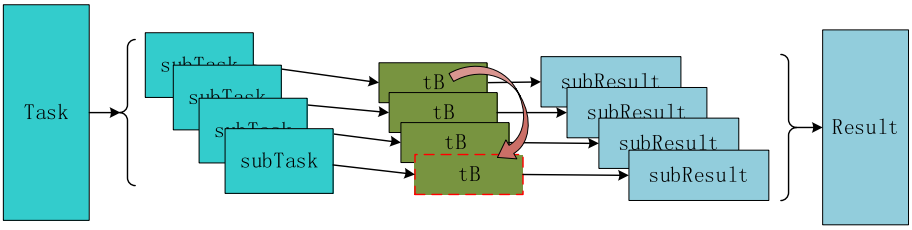

Fig. 2. Multi-thread parallel model of CUDA.

The Multi-thread parallel model divides the computing task into a set of subtasks on the granularity of threads. It should be noted that the computing task is not divided into subtasks spatially, but automatically divided by thread blocks logically. In other words, part of the whole computing task that executed by a single thread block is called subtask. Generally, a single thread block can handle one or more subtasks. When the current subtask is completed, the next subtask will be processed by the released thread block according to the predefined steps. Finally, when all subtasks are completed, the

calculation sub-results will be combined to obtain the final result. The procedures can be presented as the following equation.

$$Rt(n, t) = \sum_{i=0}^{n} K_t(subT_i) \tag{1}$$

Where n is the total number of subtasks, $subT_i$ stands for the subtask i, t represents the thread block (according to the previous analysis, we can assume that the thread blocks and subtasks correspond to each other in this paper), and K_t represents the kernel function executed by thread block t. In addition, the sub-result of $subT_i$ is represented by $K_t(subT_i)$, and Rt represents the final result of the task. From the multi-thread parallel model, it can be easily known that the computing task is divided into multiple subtasks logically and executed concurrently by the thread blocks, which can obtain a better performance theoretically in most cases. And the total time consumption of the computing task executed by multi-thread parallel model is calculated as the maximum difference between the earliest start time and the latest end time of all these subtasks handled by thread blocks. The computing method can be simply presented as the following equation.

$$T_{total}(n, t) = max\{K_t(subT_i)\} - min\{K_t(subT_i)\} \tag{2}$$

Where, as demonstrated in Eq. (1), $i = (1,\ldots,n)$, n is the number of subtasks and $subT_i$ stands for the subtask i. $K_i(subT_i)$ represents the processing procedures and the corresponding sub-result of $subT_i$ executed by thread block t. Additionally, the latest end time of all thread blocks is obtained by function *max*, and the earliest one is obtained by function *min*.

However, although the multi-thread parallel model has effectively achieved acceleration effects, the time consumption of data transfers between the host and device cannot be hidden. And computing resources are in an idle state when performing the operations of data transfer. Therefore, the more the operations of data transfer, the lower the occupation ratio of computing resource will be. To address this issue, the multi-stream parallel model was utilized.

2.2.2 Multi-stream Parallel Model

A stream in CUDA is a sequence of operations execute on device in order which are issued by the host code [13, 15, 16]. Operations within a single stream are guaranteed to execute in the prescribed order, but operations in separate streams can be interleaved and, when possible, they can even execute concurrently [17]. In other words, CUDA streams act as independent work queues through which different kernels can be executed simultaneously [18]. And, CUDA streams is a technique that can overlap kernel execution with data transfers, and different kernel executions can even be overlapped through Hyper-Q [19–21]. With CUDA streams and the feature of Hyper-Q, the overall parallelism is increased due to the added granularity of streams on the basis of threads. In fact, the multi-stream parallel model contains the multi-thread parallel model and improves the degree of parallelism (as shown in Fig. 3). When there is only a default stream, the multi-stream parallel model is the same as the multi-thread parallel model.

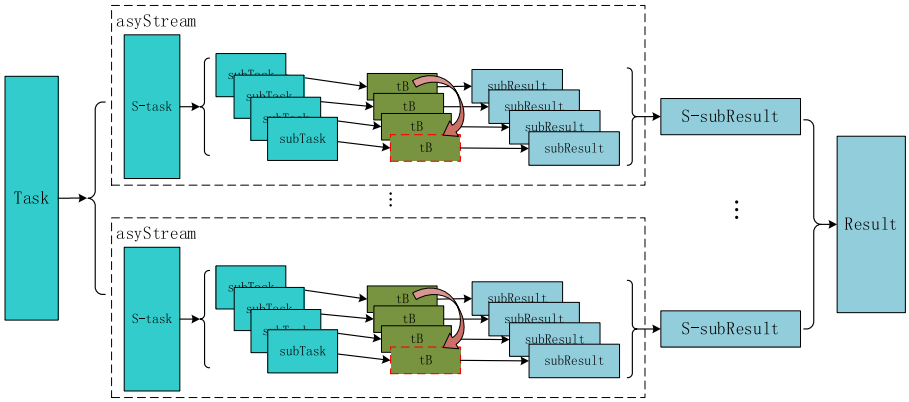

Fig. 3. Multi-stream parallel model of CUDA.

The process procedures of multi-stream parallel model can be concluded in three stages. Firstly, the multi-stream parallel model divides the computing task into subtasks (*S-task* in Fig. 3) spatially on the granularity of streams. For example, the processing task of a large size image can be divided into multiple image blocks spatially with regular size. Each of the blocks is a part of the task and then assigned to separate streams for execution. In terms of batch processing of multiple small size images, each image is generally viewed as a single *S-task* and executed on separate streams respectively. Secondly, separate *S-tasks* is executed through the multi-thread parallel model. But the difference is that the execution result of each *S-task* is only the result of its own stream (*asySresult* in Fig. 3), which is a sub-result of the whole task. Finally, in order to get the final result, sub-results will be combined in two steps: one for the multi-thread parallel model and the other for the multi-stream parallel model. The procedures can be presented as the following equation.

$$Rs(n, s, t) = \sum_{i=0}^{s} \left(\sum_{j=0}^{n} K_t(subS_iT_j) \right) \tag{3}$$

Where n is the number of subtasks on separate stream, and s is the number of streams. $SubS_iT_j$ stands for the subtask on the granularity of threads. K_t represents the kernel executed by thread block j on stream i, and K_t $(subS_iT_j)$ represents the processing procedures and the corresponding sub-result. Rs is the final result of the whole computing task. Additionally, the total time consumption of task executed by the multi-stream parallel model is calculated as the maximum difference between the earliest start time and the latest end time of all these subtasks. With reference to Eq. (2), the computing method can be formulated as Eq. (4).

$$T_{total}(n, s, t) = max\{K_t(subS_iT_j)\} - min\{K_t(subS_iT_j)\} \tag{4}$$

Although the multi-stream parallel model is commonly better than the multi-thread parallel model in terms of parallelism and performance under most cases, the complexity of designing and programming is relatively higher than that of the multi-thread parallel

model. Therefore, how to take advantages of the multi-stream parallel model more conveniently and effectively is awfully challenging.

2.3 Problem Statement

As described in the section above, it's obvious that the multi-stream parallel model can indirectly hide partial time consumption of data transfers and obtain a higher performance than the multi-thread parallel model due to the characteristic of overlap. However, CUDA streams are a series of orders executed in sequence, but different stream can execute in their own order at the same time or not regardless of sequence [10]. Paper [22, 23] have effectively improved the performance using CUDA streams. Therefore, it seems that the larger the number of streams, the greater the performance improvement. But, unfortunately, due to the constraints of hardware resources, the larger number of streams does not necessarily mean the better performance. Only the supported streams which have enough computing resources can be executed simultaneously. And the more the streams are created, the more the intervals between operations of data transfer and kernel execution are generated. In other words, it is no use to simply create more streams, because the unsupported streams will still be executed serialized at hardware level partially, as illustrated in Fig. 4, and compromise the performance of processing.

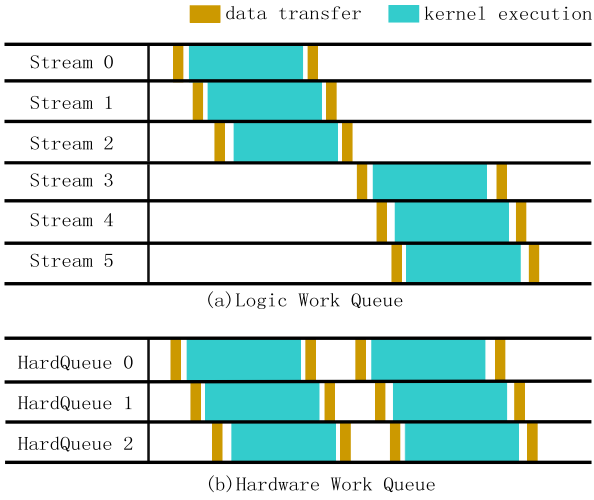

Fig. 4. Streams that executed serially partially.

The performance of device cannot be fully utilized when the number of CUDA streams is insufficient. However, the large amount of CUDA streams will cause additional time consumption that could have been avoided. Therefore, how to determine the appropriate number of CUDA streams and take fully advantage of GPU to accelerate the remote sensing image processing are awfully challenging. In order to obtain a better performance and simplify the configuration complexity of CUDA streams, this study proposes a dynamic method to overcome the problem above.

3 Algorithm Design of DAA Method

From those characteristics of GPU described above, it can be known that the GPU is effective in accelerating the remote sensing image processing and the performance can be improved significantly. To obtain the best speedup, it is suggested to design and optimize the parallel applications carefully with the basic knowledge of GPU hardware architecture and different features of the remote sensing image algorithms.

In order to obtain better performance and to use the multi-stream parallel model easily and efficiently, a dynamic acceleration method for remote sensing image processing is proposed in this section. With this method, the utilization of multi-stream parallel model can be more conveniently and efficiently applied to remote sensing image processing algorithms to a certain degree.

3.1 Adaptive Strategy of CUDA Streams

There are various means having been used to examine methods for improving concurrency and parallelism of algorithms implemented by CUDA [15, 20]. The commonly and effectively method is to optimize applications using the multi-stream parallel model on GPU. To obtain better performance, the larger size image was divided into blocks of regular size for processing in existing studies [10, 24]. From the point of view of computing task, the larger task is divided by concurrent streams for execution into multiple subtasks, as shown in Fig. 5(a) to (c). Moreover, in terms of the batch processing of small size images, the simplest method is to create an exclusive stream for each image, and effective improvement can be obtained in most cases. But, when the number of streams is too large, as analyzed above, part of the streams will be executed serially at hardware level. And as shown in Fig. 5(b), the more the streams are executed serially, the more the intervals between operations will be generated. To address this problem, the efficient approach that combine streams executed serially on the same hardware queue into one is proposed in this section. And, as shown in Fig. 5(d), intervals between the scattered streams which mapped to the same hardware queue can be reduced with stream combination.

Therefore, in order to improve performance and parallelism of applications based on GPU, the task should be divided into subtasks of appropriate size according to the various situations of computing resource, and the appropriate number of streams should be created to handle these subtasks. In terms of remote sensing image processing, there are two cases of images to be processed frequently: the processing of a single image of larger size and the batch processing of images of small size.

For the former case, in order to overlap data transfers with kernel execution, a larger size image is usually divided into blocks and optimized for the multi-stream parallel model. For the latter case, in order to avoid the situation shown as Fig. 5(b), a new method that combine input images partially before processing and creates streams adaptively, instead of creating a stream for each image, according to the hardware parameters is proposed. And, for the benefit of the adaptive method, an adaptive strategy for determining the appropriate number of streams and the size of image blocks adaptively is defined as follows:

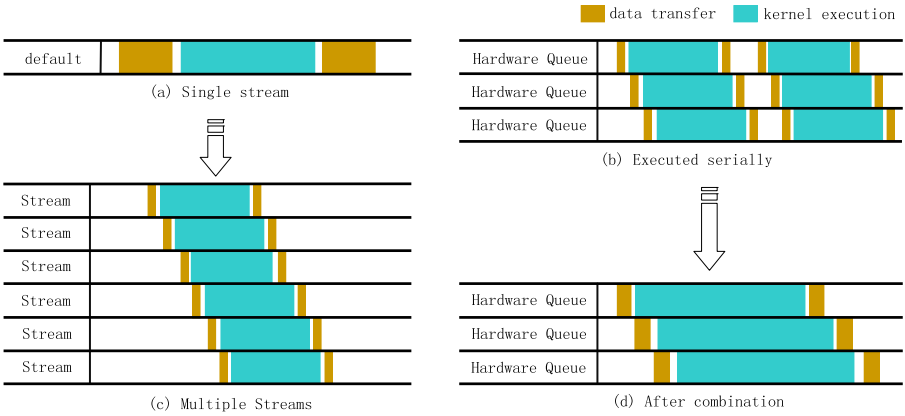

Fig. 5. Diagram of execution based on CUDA streams at hardware level.

1. The number of streams should be equal to or slightly bigger than the number of SMs, and the computing resources (i.e. threads and memories) occupied by separate stream should not exceed the resources owned by a single SM.
2. The number of image blocks should be equal to the number of streams or an integer times of the number, and the size of single image blocks should satisfy the following formula:

$$(B_s + R_s) * S_n \leq G_m \tag{5}$$

While B_s is the size of individual image block, R_s represents the size of the processing result of the image block. S_n represents the number of streams, and G_m is the total size of graphic memories.

3.2 Dynamic Adaptive Acceleration Method

Complying with these constraints defined in the previous section, an appropriate number of CUDA streams can be created, and the input images can be processed dynamically into regular-size blocks. And thus image blocks can be distributed evenly to each stream for execution, the intervals will be reduced, and the resources occupancy ratio of GPU will be increased. Furthermore, according to the analyses and strategies proposed above, a dynamic adaptive acceleration (DAA) method to divide or combine input images and to create streams dynamically for remote sensing image processing is proposed.

As shown in Fig. 6, the input images are automatically divided or combined into blocks of regular size according to the different types of them and GPU hardware parameters firstly. And then, the streams of appropriate number are created and the image blocks are loaded onto the corresponding streams for execution. Finally, execution results are combined to obtain the final result. With this method, the appropriate number of streams and image blocks can be determined automatically, and the complexity of using the multi-stream parallel model can be greatly reduced or even hidden without performance improvement declining.

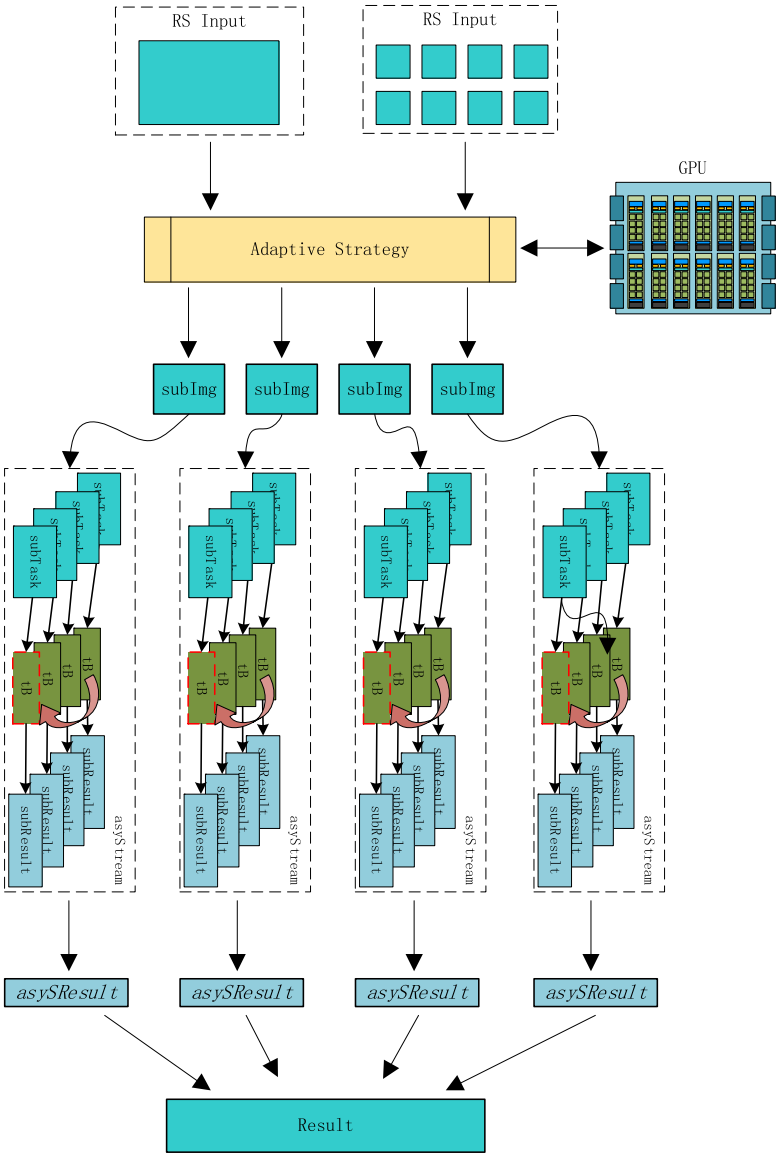

Fig. 6. Dynamic adaptive acceleration method for remote sensing image processing.

Given constraints as following, (1) the output data (remote sensing images in this paper) and the input data are roughly equal in size. (2) the time consumption of the kernel function with the same amount of data to be processed is fixed. The time consumption equations of the proposed DAA method and the general method based on multi-thread parallel model can be described as Eq. (6) and Eq. (7) respectively, in terms of the single

large size image processing.

$$2(T_{\text{int}} + \frac{1}{n}T_{copy}) + KT_{init} + KT_{launch} + KT_{compute} \tag{6}$$

$$2(T_{init} + T_{copy}) + KT_{init} + KT_{launch} + KT_{compute} \tag{7}$$

Where n represents the number of CUDA streams which determined based on the adaptive strategy, T_{int} is the time consumption and of data engine initialization and T_{copy} is the time consumption of data transfer. KT_{init}, KT_{launch} and $KT_{compute}$ represents the time consumption of kernel function initialization, kernel launch and kernel execution respectively.

Moreover, the time consumption equations of the proposed DAA method and the general method based on multi-stream parallel model can be described as Eq. (8) and Eq. (9) respectively, in terms of the batch processing of small size images.

$$2(T_{\text{int}} + \frac{\text{m}}{n}T_{copy}) + m(KT_{init} + KT_{launch} + KT_{compute}) \tag{8}$$

$$2\left\lfloor \frac{m}{n} \right\rfloor (T_{\text{int}} + T_{copy}) + 2(m\,MOD\,n)(T_{\text{int}} + T_{copy}) + m(KT_{init} + KT_{launch} + KT_{compute}) + \left\lfloor \frac{m}{n} \right\rfloor \beta \tag{9}$$

Where m represents the number of images, the number of CUDA streams in general method as well, and the $\boldsymbol{\beta}$ in Eq. (9) is defined as below.

$$\beta = \begin{cases} 0, & \text{m} = \text{n} \\ \beta, & \text{m} \neq \text{n} \end{cases}$$

The proposed DAA method can take advantages of the CUDA streams to overlap data transfers and kernel executions, and reduce the calling intervals between operations on each stream by load tasks to appropriate number of streams. According to the analysis above, it can be easily concluded that the proposed DAA method can theoretically obtain a better performance than general methods that simply based on the multi-thread parallel model and multi-stream parallel model.

4 Experiment

In this section, experiments were conducted to demonstrate the performance and effectiveness of DAA method.

4.1 Discussion and Preparation

Operations of various remote sensing image processing algorithms can be classified roughly into three types: point operation, local operation and global operation. As shown in Fig. 7, the simplest type is the point operation that the output result only depends on the corresponding input image. For local operation, the output result are correlated with the neighborhood of the pixel in input image, such as convolution calculation and median filtering. And about global operation, the output result is associated with the whole input image which is the most complex calculation.

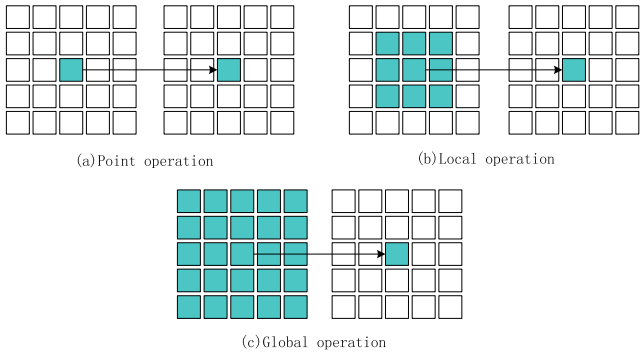

Fig. 7. Typical operation of remote sensing image processing algorithms.

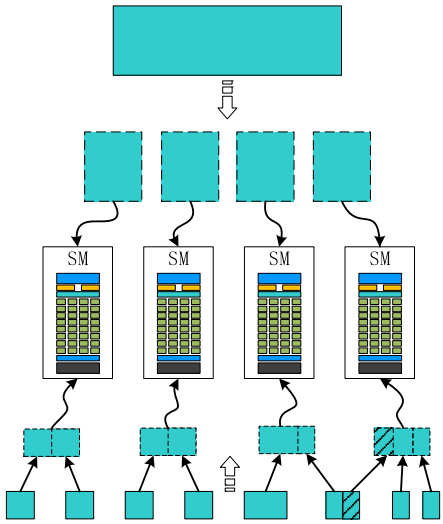

Fig. 8. Diagram of adaptive strategy for algorithms with point operations.

According to the three types of operations, it is obvious that DAA is more applicable for the algorithms with point operations because of the computational independence of point operations. Therefore, in terms of point operations, all the small-size images either of the same size or of different size can use the DAA, as shown Fig. 8.

However, when the DAA method is applied to algorithms with local operations, the adaptive strategy of DAA should be changed. In terms of large size image, the neighborhood data should be considered when DAA is applying to the algorithm of local operations. Changes should be made and vary with features of local operation algorithms, as shown in Fig. 9(a). In addition, integrity of small size images should be guaranteed, which means that small size images are required to have the same size, as shown in Fig. 9(b). Given that, design and implementation of algorithms with local or global operations accelerated by DAA method are still difficult, and the generality and

performance of DAA method will be greatly compromised. It is obvious that the DAA method is not suitable enough for algorithms with local operations or global operations. Therefore, only experiments on the algorithms of point operations was carried out in this paper.

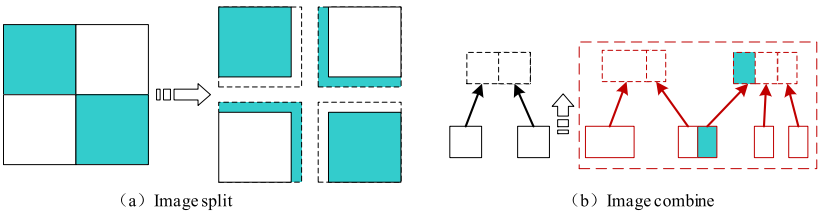

Fig. 9. Changes and constraints of adaptive strategy in DAA for local operations.

The remote sensing images that produced by GF1 satellite were selected for experiments in this section, and the hardware parameters of experimental GPU is shown in Table 1. In order to evaluate the effectiveness of the proposed method better, the time consumption of applications in host was ignored.

Table 1. Experimental environment.

GPU	GeForce GTX 1060		
Cores	1280	SMs	10
Clock rate	1.67 GHz	CUDA Cores	1280
Global memory	6 GB	UVA Support	Yes
CUDA version	9.2	Warp Size	32

4.2 Experimental Result

In the first experiment, the NDVI calculation [25], a typical remote sensing image processing algorithm of point operations is selected. The processing of large size image accelerated by DAA and general purpose multi-thread parallel model (called G-MT in this paper) was respectively discussed. To illustrate the impact of different size images, the original remote sensing images of different sizes (8540 × 8520 and 12000 × 13400) were chosen, and remote sensing images of other regular sizes (4000 × 4000, 16000 × 16000 and 20000 × 20000) were obtained by splitting and stitching. Additionally, the total size of GPU computing resources used by the method of G-MT is equal to that used by DAA. Results and perform configuration of the first experiment are shown in Table 2.

In the first experiment, the time consumption of method G-MT and method DAA was compared, including all the operations of data transfer and kernel execution. It can be easily seen from Fig. 10 that DAA had achieved better performance compared

Table 2. Experimental result.

Specifications		DAA	G-MT
Configuration	Streams	10	1 (default)
	Execution parameter	<1, 128 >	<10, 128 >
Image sizes (ms)	4000 × 4000	15.99124	25.15754
	8540 × 8520	73.2981	96.02902
	12000 × 13400	162.36974	234.44748
	16000 × 16000	253.1896	389.38958
	20000 × 20000	468.82148	651.24778

to G-MT. And with the increasing size of image, the acceleration effect was further improved. As demonstrated in previous section, the multi-stream parallel model has achieved parallelism with a higher degree than the multi-thread parallel model does. Given that, the DAA divided large size image into regular size image blocks and created streams according to the hardware situations, and then loaded image blocks evenly onto the separate stream. Therefore, DAA has obtained a better performance improvement in the processing of large size remote sensing image than G-MT. And, as the image size increases, so did the performance improvement.

In the second experiment, the application of DAA in batch processing of small size images was evaluated. There are 9 groups of images were used in this experiment, from 20 to 180 images and with an interval of 20 images between each group. In terms of image data, remote sensing tile images of 1000 × 1000 size were chosen. In this experiment, the groups of images were processed using the general purpose multi-stream parallel model (G-MS) was implemented that streams were created for each image respectively. Besides, batch processing of image groups using DAA was implemented too. Results and perform configuration of the second experiment are shown in Table 3.

As shown in Fig. 11, DAA has achieved better performance improvement than G-MS. Reasons for this improvement was analyzed in previous sections that DAA has effectively reduced the calling intervals, which were generated by G-MS frequently and increased with the number of images in batch image processing.

Particularly, when the number of images in the group is 20, DAA took the same or even more time consumption compared to G-MS. The reason is that when the size and number of images are not large enough, more streams could achieve a higher parallelism and resources occupancy rate. This is also the case for group that contains 80 images. When the number of images was 80, only slight performance improvement was achieved. However, in general, DAA has achieved better acceleration effects than G-MS

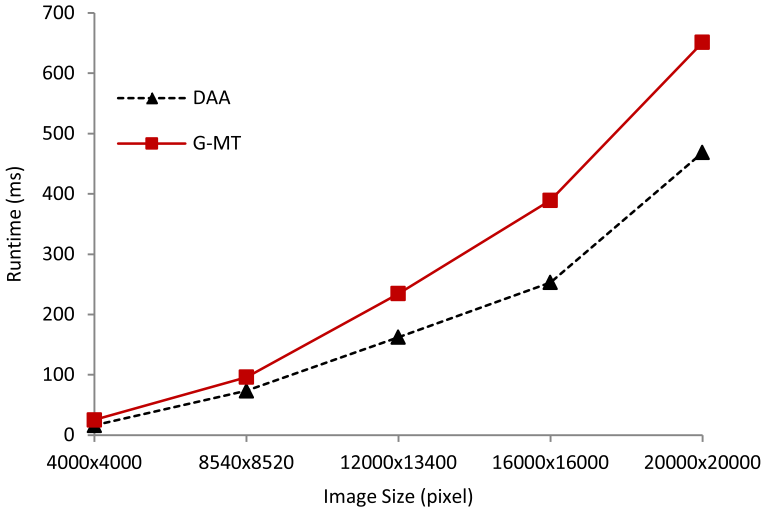

Fig. 10. Comparison of G-MT and DAA for processing of large size image.

Table 3. Experimental result.

Specifications		DAA	G-MS
Configuration	Streams	10	*Number of images*
	Execution parameter	<1, 128 >	<1, 128 >
Images	20	10.9963	10.27162
	40	16.55978	19.1056
	60	20.7585	28.35928
	80	26.70484	30.25702
	100	31.72824	39.98284
	120	36.57192	48.22432
	140	42.89492	55.39422
	160	46.34424	58.61586
	180	51.92558	65.094

in the experiment. And, as the the number of images increased, so did the performance improvement.

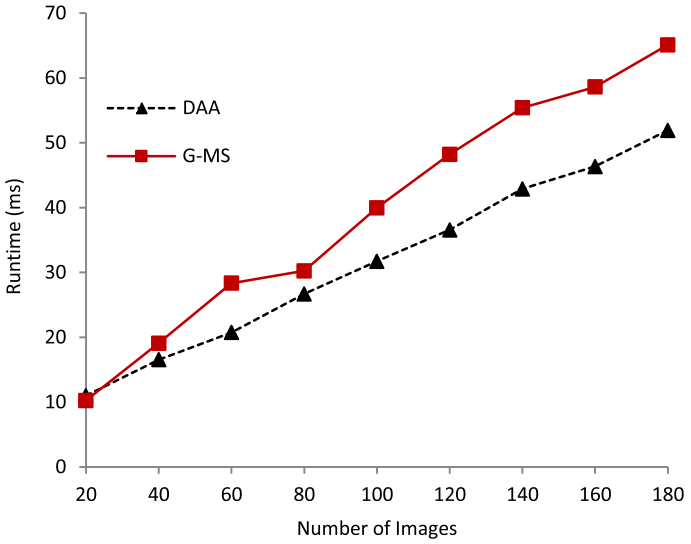

Fig. 11. Comparison of G-MS and DAA for batch processing of small size images.

5 Conclusion and Future Work

In this paper, a dynamic acceleration method for remote sensing image processing is proposed. The method can create streams and process the input images into blocks of appropriate size dynamically according to hardware parameters, and finally can load image blocks onto the corresponding streams evenly for execution. Moreover, the efficiency and performance of the proposed method in the typical remote sensing image processing algorithms of point operation is proved through experiments with different number and size of images. With this method, the two typical situations in remote sensing image processing algorithms of point operation have both obtained efficiently performance improvement, without the need to set calculation parameters of GPU and preprocess the input images manually. Therefore, it is concluded that the method proposed can be more convenient and obtain a better performance than traditional methods in accelerating remote sensing image processing.

According to the experimental results in this paper, it is obvious that the method proposed still has some deficiencies, and there was only its performance and generality in algorithms of point operation were proved. Therefore, the future study will focus on how to further improve the performance of the method proposed based on different hardware situations and computing modes [26–29]. Moreover, the application of the DAA method in other typical types of remote sensing image processing algorithms is taken into the agenda.

Acknowledgements. The authors would like to thank the referees and Editor for their helpful suggestions for revising this manuscript. The project is supported in partly by National Key Research and Development Program of China (2017YFD0301105), Natural Science Foundation of China (61202098, U1604145, U1704122), Science and Technological Research of Key

Projects of Henan Province (202102110121, 202102210352, 202102210368, 192102210096, 201400210300), and Excellent Youth Foundation of Science Technology Innovation of Henan Province (184100510004).

References

1. Giordano, R., Guccione, P.: ROI-based on-board compression for hyperspectral remote sensing images on GPU. Sensors **17**(5), 1160 (2017)
2. Gao, S., Li, L., Li, W., et al.: Constructing gazetteers from volunteered Big Geo-Data based on Hadoop. Comput. Environ. Urban Syst. **61**(b), 172–186 (2017)
3. Jiang, D., Wang, Y., Lv, Z., et al.: Big data analysis based network behavior insight of cellular networks for industry 4.0 applications. IEEE Trans. Ind. Inform. **16**(2), 1310–1320 (2020)
4. Pektürk, M.K., Ünal, M.: Performance-aware high-performance computing for remote sensing big data analytics. In: Data Mining, Chapter 5, pp. 69–90. BoD–Books on Demand (2018)
5. Levin, N., Ali, S., Crandall, D., et al.: World heritage in danger: big data and remote sensing can help protect sites in conflict zones. Glob. Environ. Chang. **55**, 97–104 (2019)
6. Ma, Y., Chen, L., Liu, P., et al.: Parallel programing templates for remote sensing image processing on GPU architectures: design and implementation. Computing **98**(1), 7–33 (2016)
7. Yusuf, A., Alawneh, S., et al.: A survey of GPU implementations for hyperspectral image classification in remote sensing **44**(5), 532–550 (2018)
8. Roui, M.B., Shekofteh, S.K., Noori, H., et al.: Efficient scheduling of streams on GPGPUs, pp. 1–33 (2020)
9. Toledo, L., Pena, A.J., Catalan, S., et al.: Tasking in Accelerators: Performance Evaluation. Parallel and Distributed Computing: Applications and Technologies (2019)
10. Hong, H., Zheng, L., Pan, S.: Computation of Gray level co-occurrence matrix based on CUDA and optimization for medical computer vision application. IEEE Access **6**, 67762–67770 (2018)
11. Xu, L., Ziedan, N.I., Niu, X., Guo, W.: Correlation acceleration in GNSS software receivers using a CUDA-enabled GPU. GPS Solutions **21**(1), 225–236 (2016). https://doi.org/10.1007/s10291-016-0516-2
12. Ikeda, K., Ino, F., Hagihara, K., et al.: An OpenACC optimizer for accelerating histogram computation on a GPU. In: 2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (2016)
13. NVIDIA: CUDA Programming Guide. https://docs.nvidia.com/cuda/archive/10.1/cuda-c-programming-guide/index.html. Accessed 28 Dec 2019
14. Wu, Z., Shi, L., Li, J., et al.: GPU parallel implementation of spatially adaptive hyperspectral image classification. IEEE J. Sel. Top. Appl. Earth Observations Remote Sens. **11**(4), 1131–1143 (2017)
15. Li, T., Narayana, V.K., El-Ghazawi, T.: Symbiotic scheduling of concurrent GPU kernels for performance and energy optimizations. In: Proceedings of the 11th ACM Conference on Computing Frontiers, p. 36. ACM, Cagliari, Italy (2014)
16. Li, W., Zhang, L., Zhang, L., et al.: GPU parallel implementation of isometric mapping for hyperspectral classification. IEEE Geosci. Remote Sens. Lett. **14**(9), 1532–1536 (2017)
17. Baca, H.A.H., Valdivia, F.D.L.P.: Efficient sparse matrix-vector multiplication on GPUs using the CSR format, pinned memory and overlap data transfer. In: 2019 IEEE XXVI International Conference on Electronics, Electrical Engineering and Computing (2019)
18. Kim, J., Cha, J., Park, J.J.K., et al.: Improving GPU multitasking efficiency using dynamic resource sharing. IEEE Comput. Archit. Lett. **18**(1), 1–5 (2019)

19. Adriaens, J.T., Compton, K., Kim, N.S., et al.: The case for GPGPU spatial multitasking. In: IEEE International Symposium on High-Performance Comp Architecture (2012)
20. Luley, R.S., Qiu, Q.: Effective utilization of CUDA Hyper-Q for improved power and performance efficiency. In: 2016 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPSW, pp. 1160–1169. IEEE, Chicago, IL (2016)
21. Dominguez, J.M., Crespo, A.J.C., Valdezbalderas, D., et al.: New multi-GPU implementation for smoothed particle hydrodynamics on heterogeneous clusters. Comput. Phys. Commun. **184**(8), 1848–1860 (2013)
22. Czarnul, P.: Benchmarking overlapping communication and computations with multiple streams for modern GPUs. Ann. Comput. Sci. Inf. Syst. **17**, 105–110 (2018)
23. Knap, M., Czarnul, P.: Performance evaluation of Unified Memory with prefetching and oversubscription for selected parallel CUDA applications on NVIDIA Pascal and Volta GPUs. J. Supercomput. **75**(11), 7625–7645 (2019). https://doi.org/10.1007/s11227-019-02966-8
24. Yang, Z., Zhu, Y., Pu, Y.: Parallel image processing based on CUDA. In: 2008 International Conference on Computer Science and Software Engineering, pp. 198–201. IEEE, Hubei, China (2008)
25. Alvarez-Cedillo, J., Herrera-Lozada, J., Rivera-Zarate, I.: Implementation strategy of NDVI algorithm with Nvidia thrust. In: Pacific-Rim Symposium on Image and Video Technology, pp. 184–193. Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-53842-1_16
26. Kiani, A., Ansari, N., et al.: Edge Computing Aware NOMA for 5G Networks. IEEE Internet Things J. **5**(2), 1299–1306 (2018)
27. Campostaberner, M., Morenomartínez, Á., Garcíaharo, F.J., et al.: Global estimation of biophysical variables from Google earth engine platform. Remote Sens. **10**(8), 1167 (2018)
28. Kumar, L., Mutanga, O., et al.: Google earth engine applications since inception: usage, trends, and potential. Remote Sens. **10**(10), 1509 (2018)
29. Gorelick, N., Hancher, M., Dixon, M., et al.: Google earth engine: planetary-scale geospatial analysis for everyone. Remote Sens. Environ. **202**, 18–27 (2017)