Personal and Ubiquitous Computing (2019) 23:465–474
https://doi.org/10.1007/s00779-019-01222-3

ORIGINAL ARTICLE

# GPU-based parallel optimization for real-time scale-invariant feature transform in binocular visual registration

Jiashen Li[1] · Yun Pan[2]

Received: 24 January 2019 / Accepted: 13 April 2019 / Published online: 30 April 2019

**Abstract**
Scale-invariant feature transform (SIFT) is one of the widely used interest point features. It has been successfully applied in various computer vision algorithms like object detection, object tracking, robotic mapping, and large-scale image retrieval. Although SIFT descriptor is highly robust towards scale and rotation variations, the high computational complexity of the SIFT algorithm inhibits its use in applications demanding real-time response and in algorithms dealing with very large-scale databases. In order to be effective for image matching process in near real-time, the Compute Unified Device Architecture (CUDA) application programming interface of a graphics processing unit (GPU) is incorporated to speed up or improve the SIFT method. Experimental results show that the proposed GPU-based SIFT framework is suitable for image application in real time. It can improve the image matching process both in time and accuracy compared with conventional SIFT method.

**Keywords** Scale-invariant feature transform · Image matching · Parallel optimization · Real-time

## 1 Introduction

Currently, a graphics processing unit (GPU) has been widely used in general-purpose computing [1]. Due to the programmability of hardware, the applications of earlier general-purpose computing are limited; the development is also very difficult. NVIDIA launched the Compute Unified Device Architecture (CUDA) [2], a parallel computing device as a data system of hardware and software. The programming style is simple, and it uses a multi-thread parallel processing and makes people take good advantage of GPU.

On the other hand, image matching is a key technology of image processing; it is the basis for the more advanced image processing technology. SIFT matching algorithm is a stable feature-based matching algorithm. SIFT has better resistance for scaling, rotation transformation, and illumination. It is a hot research field of image processing [3].

✉ Jiashen Li
ljszju@gmail.com

[1] College of Electrical Engineering, Zhejiang University, Hangzhou, China

[2] College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou, China

While SIFT matching algorithm has many advantages, it is a time-consuming operation. And since it is a matching algorithm based on feature, the number of matching points is less. The obtained matching points cannot satisfy the demand for generating a disparity map. Therefore, the applications are limited. SIFT matching algorithm exists with these two problems. On the one hand, this paper discusses the traditional preprocessing algorithms, analyzes the parallelism of the algorithms, and implements gray conversion, Gaussian filtering, histogram equalization, and Wallis filtering on CUDA [4]. And the paper improves the original SIFT algorithm, it makes matching points that get from the original algorithm as seed points, and then make regional growth, traverse the entire image, so you can get many matching points which are good for generating the disparity map. On the other hand, the paper uses the CPU and GPU heterogeneous platforms and analyzes the CUDA programming model and memory model. We divide the tasks of SIFT matching algorithm and analyze the algorithm in detail so the algorithm can be carried out on CUDA. GPU accelerates the speed of the algorithm [5].

The experiment of the regional growth algorithm based on SIFT was carried out on the platform of binocular vision; it verified the effectiveness of the improved algorithm [6]. In order to evaluate the acceleration of GPU more directly, the experiment of the SIFT matching algorithm based on GPU was carried out in a variety of different scenarios. The results show that the algorithm not only improves the speed of the algorithm on CUDA, but also ensures the stability of the original algorithm. The efficiency of the algorithm has been improved in [6], where an improved SIFT algorithm is proposed in this paper for the problem of SIFT algorithm (scale-invariant feature transform) which has high computational complexity and long running time. In order to solve this problem, the improved SIFT algorithm in this paper expands the range of extremum points to reduce the number of extreme points so as to increase the speed of operation. Furthermore, the improved algorithm uses a circular window of 12 rings instead of the traditional square window and simplifies the construction of SIFT feature descriptors by generating 78-dimensional SIFT feature descriptors, which further improve the operation speed of the algorithm. For the best experimental results, a simple yet robust method is used in this improved algorithm [7]. It is applied in the initial registration search between feature point pairs and is used to perform secondary processing on the feature point registration pairs to eliminate mismatching. Finally, this paper combines the improved SIFT algorithm with the gradual image fusion algorithm to realize the mosaic and fusion of time series images. To evaluate the experimental effect, this paper uses partial patch detection to evaluate after image matching and SIFT feature extraction. The experimental results show that the algorithm of this paper has fast computation speed, high robustness, and good fusion effect. The SIFT matching algorithm can extract feature points that have a strong resistance to the external environment, and the matching using vectors is relatively stable. However, since the algorithm itself is too complicated, and it takes a lot of time to obtain the feature point, it will be limited in practical applications. In order to speed up the process, a simple yet robust SIFT algorithm based on GPU acceleration platform is proposed for improving the running speed. The SIFT matching algorithm is optimized parallelly, and the tasks are correctly assigned on the CPU and GPU so that the SIFT matching algorithm is implemented on the CUDA platform. Our proposed parallel programming architecture is applied to the SIFT-based region matching algorithm, where many standard images are used to evaluate the efficiency of the algorithm.

## 2 Our improved SIFT algorithm

The SIFT algorithm constructs the scale space of the image to detect the extreme points so as to determine the key point [8]. And then the key point is taken as the center to get its neighborhood, which can construct the SIFT feature point descriptor. And finally, the image is registered according to the obtained feature descriptor.

### 2.1 Detection of key points in scale space

The SIFT algorithm constructs the Gaussian pyramid of the image by Gaussian function and then uses the image difference between the adjacent Gaussian scale spaces in the Gaussian pyramid to represent the response value image of the difference of Gaussian (DoG). The DoG response image can be expressed as

$$\begin{aligned} D(x,y,\sigma) &= (G(x,y,k\sigma)-G(x,y,\sigma))\otimes I(x,y) \\ &= L(x,y,k\sigma)-L(x,y,\sigma) \end{aligned} \tag{1}$$

where $I$ is the original image, $L$ is the scale space of the image, the size of $\sigma$ is denoted as the different scales, and the larger the $\sigma$ is, the more blurred the image is.

After establishing the differential Gaussian pyramid [9], the DoG local extremum points are initially detected and their spatial positions and scales are accurately located. Since the candidate extreme points for coarse search include unstable points with weak contrast and strong edge response, the curve fitting strategy is used to eliminate unstable extreme points in the candidate sets. After removing the unstable key points, the gradient direction distribution characteristics of the neighboring pixels of the remaining qualified key points are calculated to determine the direction of the key points. The direction of some key points includes the main direction and other directions, and finally the SIFT feature area can be obtained.

### 2.2 Local feature description

The calculated key points are described to obtain a local feature descriptor, which includes three parts: correcting the main direction of rotation, generating descriptors, and normalizing. After adjusting the coordinate axes according to the direction of the key points to make the two consistent, in the scale space corresponding to each key point, a square window centered on the key point and having a size of $16 \times 16$ is selected as a neighborhood range [10]. The range is divided into $4 \times 4$ sub-regions, and the gradient magnitude and direction of the pixels on each sub-region are calculated to obtain a cumulative histogram of the gradient information, and a seed point containing eight-direction vector information is generated. Thus, a key point can generate a $4 \times 4 \times 8 = 128$-dimensional SIFT feature descriptor [11]. Each feature descriptor includes not only key points, but also pixels that contribute to it around the key points. This has indicated that the image information of the area is unique. Finally, the feature descriptors are normalized to eliminate the effects of illumination changes.

The SIFT matching algorithm can extract feature points that have a strong resistance to the external environment, and the matching using vectors is relatively stable. However, since the algorithm itself is too complicated, and it takes a lot of time to obtain the feature point, it will be limited in practical applications. In order to speed up the process, a simple yet robust SIFT algorithm based on GPU acceleration platform is proposed for improving the running speed. The SIFT matching algorithm is optimized parallelly, and the tasks are correctly assigned on the CPU and GPU so that the SIFT matching algorithm is implemented on the CUDA platform [12]. Our proposed parallel programming architecture is applied to the SIFT-based region matching algorithm, where many standard images are used to evaluate the efficiency of the algorithm.

## 3 GPU-based heterogeneous architecture for SIFT

The so-called heterogeneous computing refers to the collaborative computing of CPU + GPU or CPU + other devices (such as FPGA [13–16]). Generally, our program is calculated on the CPU. However, when a large amount of data needs to be calculated, the CPU seems unable to solve it. Therefore, heterogeneous computing can be adopted to improve calculation speed. For example, the computing power of a computing device such as a CPU (central processing unit), a GPU (graphics processing unit) [17], or even an APU (Accelerated Processing Units) [18] can be utilized to increase the speed of the system. Heterogeneous systems are becoming more common and are receiving increasing attention for computing [19].

Recently, the most used heterogeneous computing is to use the GPU to accelerate [20]. The mainstream GPUs all use a unified architecture unit. With a powerful programmable stream processor, the GPU is far behind the CPU in single-precision floating-point operations [21]. The complete process based on GPU architecture is shown in Fig. 1.

Firstly, the data is initialized on the CPU, and then the task is divided; the task suitable for GPU parallel is sent to the GPU for processing. Therefore, the parallel analysis and implementation on the CUDA platform are achieved, and finally the calculated result is transmitted back to the CPU, where the final result is generated.

According to the above process, a parallel analysis of the SIFT matching algorithm is implemented on the CUDA platform, which can accelerate the algorithm performance. It is necessary to divide the algorithm into correct tasks. Different task partitioning will produce different acceleration effects, and the wrong task partitioning may even have counter-effects, so it is very important to analyze the parallelization of the algorithm [22].

In the SIFT algorithm, tasks with strong logic or less computation are placed on the CPU. For example, image input, image initialization, etc. can be placed on the CPU for processing, and the Gaussian pyramid establishment and key point extraction are placed on the GPU. The specific process is to input an image on the CPU and calculate the Gaussian kernel function data and pass the obtained convolution template to the GPU constant memory, and in addition, the image data is delivered to the GPU to establish a Gaussian pyramid. Then, the adjacent image layers are subtracted in the CPU so as to obtain a differential pyramid. The feature points are detected in the differential scale space; the position and scale information of the detected feature points are loaded into the GPU. Thus, the key points are accurately located in the GPU, and the main direction of the key points is extracted, and the key point information is passed back to the CPU, then extract the SIFT feature, and finally complete the SIFT feature matching in the GPU.

In the process of completing the assignment task, the SIFT algorithm should make full use of the advantages of the GPU to optimize the parallel process. In general, the management of stream can play the role of optimization algorithm. In CUDA, a stream is an operation sequence. However, when different streams are used for data transmission, they can perform parallel calculations at the same time. Stream processing can reduce the delay in data processing between the host and the device, which needs to use page-locked to store data on the host to perform asynchronous transmission. The general

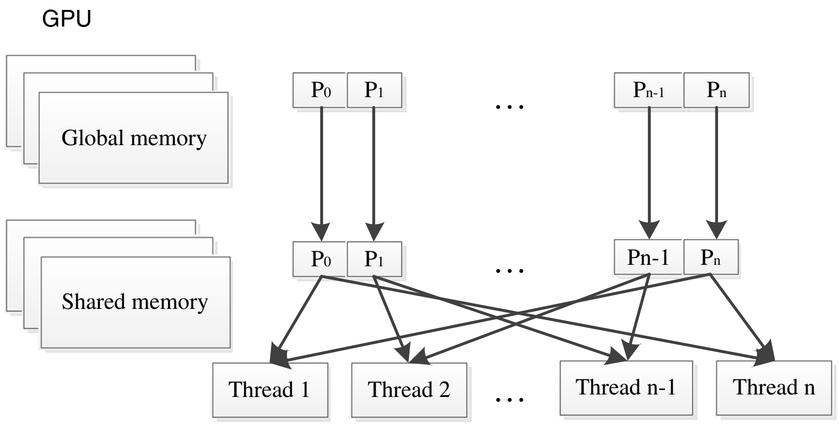

**Fig. 1** Heterogeneous model based on GPU architecture

parallel tasks are divided into three steps: transferring data from the host to the device, performing parallel processing on the core function in the device, and displaying the data from the device to the host. The data is sent back from the device to the host, and there is no overlap. In parallel execution, the system can perform parallel processing on the device while performing data transmission. When one stream is transmitting data, the other stream can be processed in parallel, which fully utilizes the performance of the GPU.

Optimizing memory access also can speed up performance. Each memory has its own characteristics. It combines the memory with the data format and selects the appropriate memory to store the data. Shared memory has a very fast access speed, and some intermediate data can be stored in it, which can reduce access to global memory with relatively slow access speed. However, if the storage space of the shared memory is limited, it is unrealistic to store all the data of the image in the shared memory. The shared memory can be used to store some data stored in the global memory, and then processed them. The specific use is shown in Fig. 2. The constant memory is a read-only memory that can be accelerated by the cache to store constant data that is constant and to be used frequently. The mapping is similar to shared memory.

## 4 Parallel analysis for feature point detection

### 4.1 Construction of scale space

The construction of scale space is the basis of the whole SIFT algorithm, where there are a lot of data calculations. In order to build scale space, the steps of the SIFT algorithm are described as follows:

(1). The Gauss pyramid is grouped into $t$ groups, and $s$ is defined as the number of layers between$\sigma$ and $2\sigma$. In order to achieve continuous scale changes, each group should be divided into $s + 3$ layers. This is because when comparing the extreme values, the first and last two layers of each group of images cannot be compared, so it is necessary to continue Gaussian blurring on the top layer of each group to generate three images.

(2). Calculate the Gaussian kernel of each layer of the Gaussian pyramid.

(3). The second layer of images can be obtained by convolution in the first layer, and the entire set of Gaussian pyramids can be obtained by analogy.

(4). The image obtained by down-sampling the middle layer image in each group of images is used as the first layer image in the next group of images, and then the previous steps are repeated to obtain the entire image of the next group. This is done in turn, and finally a complete Gaussian pyramid can be constructed.

(5). The adjacent layers are subtracted in each group to generate a DoG space.

Step 1 and step 2 belong to the data preparation required for the construction of the scale space; the parallel degree is not high, and the amount of data is not large, so they can be placed on the host for calculation. Steps 3 and 4 are a large amount of parallel data for processing; this part is placed on the co-processor, namely GPU so as to obtain efficient run-time performance. The whole Gauss pyramid can be generated after $t \times (s + 3)$ cycles in the kernel function, and a layer of Gauss pyramid image can be generated when the kernel function is executed once. The generation of the differential Gauss pyramid and the generation of the Gauss pyramid are simultaneous. For each layer of the image generated by the kernel function, it is subtracted from the adjacent layer of the image so that a layer of differential pyramid image can be generated.

In the process of image convolution, we adopt the convolution method based on row-column separation. Thread

**Fig. 2** Data multiplex for GPU

patches are set to two-dimensional and threads are one-dimensional. We use blocks to manage the row data and subdivide the pixels into threads to calculate. We use shared memory and texture memory to optimize the parallel degree. The task of the differential pyramid is relatively simple. Each patch calculates the data of a rectangular area in the image, and each thread calculates the difference between the corresponding pixels of the two adjacent layers of the image. The calculated results are stored in the DoG memory area.

### 4.2 Parallel analysis for extreme point detection

In order to determine if a pixel is an extreme point in SIFT, the steps are shown as follows:

(1) If the value of each pixel in differential Gaussian space is greater than or less than the eight pixels around it, the next step will be taken.
(2) Comparing the remaining points with the nine adjacent pixels in the upper and lower layers, if the value is greater or less than the pixels, the next step will be taken.
(3) After five times of interpolation, the position of the extremum is accurately located to sub-pixel level, and the contrast after interpolation is judged. If the value is smaller than the threshold, this point will be removed.
(4) The edge points are judged by the Hessian matrix.

The detection of the extreme points between the groups is performed separately, because the pixels of the images between the different groups are different and independent of each other. The loop for *t* times a kernel function is executed to complete the detection of extreme points in a set of image data. In this paper, we do not directly compare the pixel points with the 26 pixels around but compare them with the 8 pixels of the same layer. After removing most of the unqualified points, it is compared with the adjacent points in the upper and lower layers, which greatly speeds up the efficiency of the algorithm. Two-dimensional block and one-dimensional thread are adopted to process the detection of extreme points. Given each block to be responsible for $16 \times 16$ pixels, so for an image with the size of $width \times height$, we need $(width + 1) \times (height + 1)$ blocks if its width and height cannot be divisible by 16. So block in the calculation to remove the remaining data that can be divisible by 16, not only needs cross-border judgment, warp also make branch judgments. In order to improve the result, if the width and height of the image cannot be divisible by 16, the zero element is added to the outer edge of the image, so that the width and height of the image can be divisible by 16, which can reduce the branch judgment of the warp.

In the implementation of the kernel function, the definition of two-dimensional block contains 256 threads, and each thread processes one pixel. Therefore, the shared memory will be used to speed up the storage of data. The data in each patch can be communicated through shared memory, and the image data of the DOG space calculated in the patch is stored in the shared memory, which can be used when calculating the main curvature and contrast. As you can see, these operations can improve the efficiency of the program. In addition, the use of stream can improve the use of the GPU when the edge points are removed.

### 4.3 Calculation of feature point gradient

When calculating the gradient of the feature points, the required data has the data of the detected feature points and the pixel points around the feature points. Since the scale and position of the feature points are not fixed, the actual input is the data of the differential pyramid space; the detailed calculation steps are as follows:

(1) The range of histogram is divided into 10 directions, and the modulus and direction of the pixels in the range of R are calculated around the radius of the feature points.
(2) Calculate the Gauss weights of points in the range with R radius, transform 10 directions into array indexes from 1 to 10, and then add the product of Gauss weights and modulus to the corresponding values of the index.
(3) The direction of maximum modulus in the histogram is taken as the main direction, and the direction of 80% peak value is added as the secondary direction to the feature point array.

As can be seen from the above analysis, the task of calculating the feature point modulus and direction can be placed on the GPU, and the task of searching the main and secondary directions is placed on the host. If the task of finding the direction is also placed on the GPU, the branch of the warp will be added again and using the _syncthreads() for synchronization, which will affect the speed of the kernel function. There are only 10 directions in S, and the calculation is not complicated. It is allowed to run on the CPU.

Considering that each patch supports up to 1024 threads for parallel processing, we take *r* as 15. In fact, the farther away from the feature points, the smaller the influence of the pixels on the feature points, so the calculation of the main direction with $r = 15$ has little influence.

### 4.4 Feature descriptor calculation

The calculation of the feature descriptor is similar to the calculation of the feature point gradient. The necessary input data in the program has an array of feature points and differential pyramid data.

For a $16 \times 16$ area, it is divided into 16 small areas, and each small area generates an 8-dimensional vector, so that the feature point descriptor of the 128-dimensional vector can be used to obtain the invariance of the brightness change through normalization. When processing on the GPU, a thread patch is used to process a 128-dimensional vector of a feature point, and each thread corresponds to a sample point; in other words, a contribution of a seed point formed in a small area to a feature point can be processed. The obtained results are added to each direction, and finally the entire grid can obtain a 128-dimensional vector of all feature points. Normalization is placed on the CPU that handles logical transaction processing.

### 4.5 Parallel analysis of feature matching

The more time-consuming in SIFT feature matching is the search work on the KD-tree. In our proposed architecture, KD-tree is built on the CPU, and the work on the GPU is to find the two nearest neighbors. Given a thread patch to contain 128 threads and if we use a thread to search for a feature point, then a patch can search 128 feature points.

When the number of feature points is less than 128, the zero-padding operation can be performed according to the described method above, which satisfies the minimum calculation requirement of one thread patch, and reduces the overhead for making judgments. When the number of feature points is much larger than 128, parallel calculation is performed directly with multiple thread patches. After the KD-tree is built on the CPU, the texture memory cache is used here to bind the KD-tree to the texture memory, which can greatly improve the data access speed. When searching for the nearest neighbor on the GPU, the data information found is stored in the shared memory of each thread patch, which makes reading the data more rapidly.

When searching for the nearest neighbor, the Euclidean distance is used to calculate the similarity between them. When calculating the Euclidean distance between two points on the GPU, it is assumed that the two key points stored in the global memory are represented by eight-dimensional vectors, which are $a$ vector $(a_0, a_1, \cdots, a_7)$ and $b$ vector$(b_0, b_1, \cdots, b_7)$ respectively. The square value of the difference calculated for each dimension is stored in the shared memory, and then the square root of the result is obtained by three times accumulation operation. Thus, the Euclidean distance between $a$ and $b$ vectors is obtained.

## 5 Experiment results and analysis

The CUDA and CPU experiments in this paper are running under the Ubuntu 16.04 LTS environment using the Terrans Force X411 computer which is configured for Intel Core i7-6700HQ, RAM 16GB, NVIDIA GeForce GTX 1050Ti; the embedded experiments used the NVIDIA TX2 development board. Software development kits such as NVIDIA CUDA Toolkit 8.0 and OpenCV 3.2.0 are used in the experiment in this paper. We use SIFT+ GPU to capture real-time collection and record the matching time of each stage. Experiments show that the SIFT + GPU algorithm proposed in this paper has a distinguish performance improvement in the feature extraction and feature matching, whose speed is 639 times of classic SIFT algorithm.

### 5.1 Evaluation index

In order to verify the efficiency of the GPU-based SIFT algorithm proposed in this paper, we will test the standard images provided by the benchmark database. These standard images are all corrected. By comparing the feature points obtained by our proposed algorithm with those of traditional SIFT methods, we can evaluate the effectiveness of the algorithm more accurately. The commonly used evaluation indicates the percentage of mismatched pixels. The expression is written as follows:

$$e = \frac{1}{N}\sum(|d_c(x,y) - d_T(x,y)| \geq \delta_d) \quad (2)$$

where $d_c(x, y)$ is the disparity of the calculated disparity map at the point $(x, y)$, $d_T(x, y)$ is the disparity of the real disparity map of the image at the point, $N$ is the sum of pixels in images, and$\delta_d$ is the threshold of error detection, which is generally equal to 1.

### 5.2 Qualitative and quantitative performance analysis

In order to better analyze the experimental results, six groups of images with different size are selected for experiments. The size of six groups of image pairs to be matched are 256 × 320, 512 × 640, 720 × 540, 1024 × 768, 1080 × 810, and 1080 × 1920, respectively. Since the matched images are the same and the process is similar, the image matched effect on GPU and CPU is almost the same, but the efficiency of image matching is different. Therefore, instead of repeating the effect image in the process of matching based on CPU and CPU+GPU platform, we focus more on comparing the speed of image matching. Due to space limitation, only a set of experimental results are given here.

The resolution of the two images is 1080 × 1920 in Fig. 3 where the image of (a) and (b) is from the benchmark database. Figure 3c is the registered points for rough matching in the basis of SIFT feature points. From the figure, it can be seen that there are still some mismatching pairs among many feature points, which will have a great impact on the accuracy of the calculation results of the transformation matrix of the two images. Therefore, precise matching of feature points should be carried out. Figure 3 d is an image obtained by precise registration of the calculated matching points by iteration using GPU matching algorithm. It can be seen that the matching of feature points in the overlapping area of the two images is basically correct, and there is no obvious mismatch. Since the results of CPU and GPU are similar, the validity of GPU is fully illustrated.

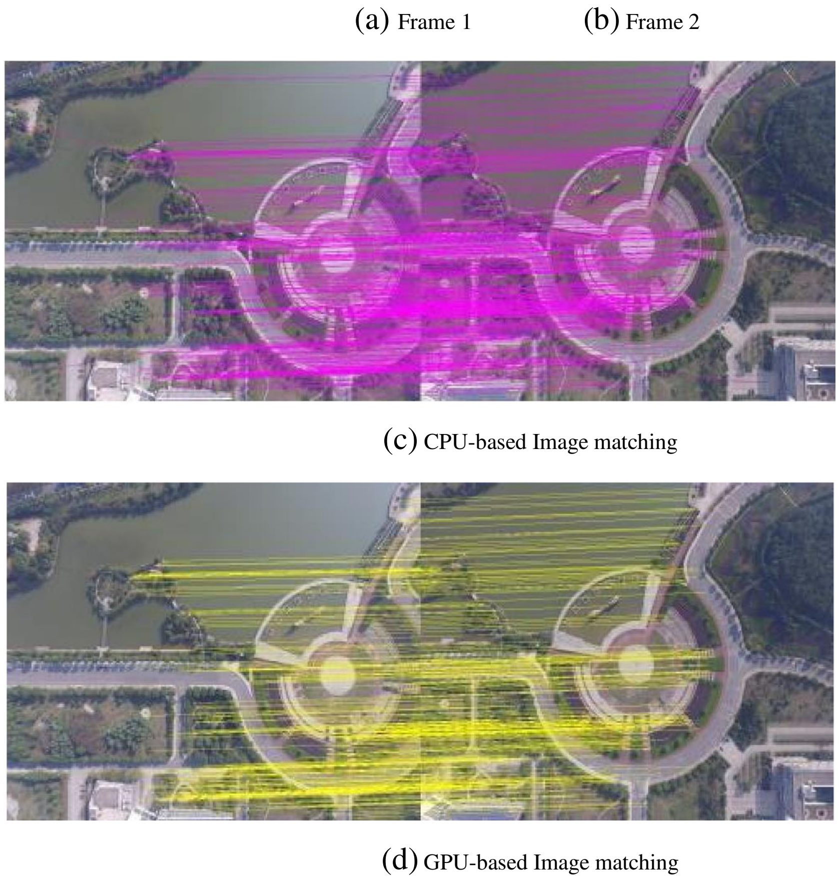

**Fig. 3** Comparison results of image matching on CPU and GPU. **a** Frame 1. **b** Frame 2. **c** CPU-based image matching. **d** GPU-based image matching

The data in Table 1 is the time and the number of feature points taken by the SIFT operator to execute on the CPU serial and the GPU parallel, respectively. In addition, the acceleration ratio is adopted to verify the effectiveness of our architecture. In order to avoid the random error caused by computer cache and other factors, this paper executes the SIFT matching in each group of images on the CPU and CPU + GPU heterogeneous platform for 10 times and then obtains the average value as evaluation index, using the average value as the table. The data value of each item.

From the number of feature points of the six groups of images in Table 2, the SIFT algorithm performs the same number of feature points on the CPU and GPU respectively. In other words, the results of the SIFT algorithm executing the extracted feature points on CPU and GPU are consistent. From the perspective of algorithm performance, the time taken by the SIFT operator in Table 2 to execute on the CPU serial and GPU parallel is related to the amount of computation. From the image itself, the time spent by the SIFT algorithm is mainly related to the resolution of the image and the number of extreme points. The greater the resolution of the image is, the longer the data transmission time between the CPU and the GPU is. In the process of DOG pyramid construction, it is directly operated with each pixel of the image, and the more pixels in the image, the more time it will take in this respect; in addition, the more extreme points in the image, the more time it will take to eliminate extreme points and construct feature descriptors.

The experimental data of group E and group F in Table 2 have the same image resolution, but they contain a different number of feature points. The more feature points extracted from two images in group F, the more time it takes and the greater the acceleration ratio. Such results also verify that the more data the GPU calculates, the more obvious the parallel acceleration effect is.

In the six groups of experiments, the two images in group A have the lowest resolution and the least feature points. In the SIFT parallel computing, the acceleration effect is the worst, but the acceleration ratio is still 1.836. Therefore, the parallel

**Table 1** Performance comparison for image matching on CPU and GPU

| Images | CPU-based image matching | GPU-based image matching |
|---|---|---|
| A (256 × 320) | 3 | 2 |
| B (512 × 640) | 9 | 9 |
| C (720 × 512) | 8 | 7 |
| D (1024 × 768) | 9 | 8 |
| E (1080 × 810) | 7 | 7 |
| F (1080 × 1920) | 8 | 8 |

**Table 2** Comparison of SIFT feature points extraction on CPU and GPU

| Images | SIFT-CPU | | | SIFT-GPU | | | |
|---|---|---|---|---|---|---|---|
| | Number of feature point | | Times | Number of feature point | | Times | Acceleration ratio |
| A (256 × 320) | 466 | 367 | 1.579 | 440 | 375 | 0.860 | 1.863 |
| B (512 × 640) | 1478 | 1761 | 6.367 | 1546 | 1765 | 1.798 | 3.540 |
| C (720 × 512) | 2413 | 2367 | 9.554 | 2406 | 2354 | 1.930 | 4.925 |
| D (1024 × 768) | 4119 | 4279 | 17.503 | 4180 | 4270 | 2.245 | 6.558 |
| E (1080 × 810) | 3691 | 3412 | 16.791 | 3691 | 3480 | 2.110 | 6.452 |
| F (1080 × 1920) | 3412 | 5218 | 18.764 | 3398 | 5218 | 2.606 | 7.2 |

design of the SIFT algorithm is feasible in this paper. The two images in group F have the best acceleration effect, reaching 7.12. As can be seen from Table 2, the resolution of these two images is the highest, and the feature points in the image are the most. Since GPU is good at dense data computing, the acceleration effect of GPU-based SIFT feature point extraction is also the most obvious. For the D and E experiments, the image resolutions are different. Although the image size of the D group is slightly smaller than that of the F group, the images in the D group contain more feature points than the E group, so the acceleration effect of the group D is better than that of the group E.

Table 3 shows the time spent on matching images in CPU and GPU in six groups of experiments. In combination with the number and time consumed by the SIFT algorithm in Table 3 for feature points extraction on CPU and GPU respectively, the acceleration effect of group A is poor. Firstly, the image is relatively small, and the feature points are also the least, so the calculation amount is also small, so the acceleration effect on the GPU is not obvious. The best time to accelerate the image matching is the group E. It can be seen from Table 3 that the acceleration effect of SIFT feature points extraction from two images on GPU is not the best, but it contains fewer feature points than group D and group F, so the time of feature points matching in the latter is less than group D and group F.

**Table 3** Comparison for image matching on CPU and GPU

| Images | CPU-based image matching | GPU-based image matching | Acceleration ratio |
|---|---|---|---|
| A (256 × 320) | 3.325 | 2.701 | 1.25 |
| B (512 × 640) | 14.524 | 9.524 | 1.462 |
| C (720 × 512) | 15.161 | 7.253 | 2.018 |
| D (1024 × 768) | 21.995 | 8.155 | 2.685 |
| E (1080 × 810) | 24.275 | 7.652 | 3.295 |
| F (1080 × 1920) | 23.162 | 8.132 | 2.819 |

Fig. 4 and Fig. 5 are time comparison curves of SIFT algorithm and image matching on CPU and GPU, respectively. It can be seen that when the image pixels are small, the GPU acceleration effect is not obvious. This is because when extracting feature points from the image with small pixels, the data of the image is relatively small, and the time of acceleration on GPU is relatively small. Then the time of data uploading from CPU to GPU and the time of data returning from GPU to CPU offset a large part of the time of acceleration in GPU parallel computing. When the pixels of the image increase, the acceleration effect of GPU becomes more and more obvious. Although the time of data information uploading to GPU and information returning from GPU also increases, the parallel computing acceleration effect is better, so it can better hide the part of the time that the image increases when transmitting data. However, when the pixel value of the image is large, it will be limited by the bandwidth, which makes the data transmission time between CPU main memory and GPU display memory greatly weaken the acceleration effect of GPU parallel computing.

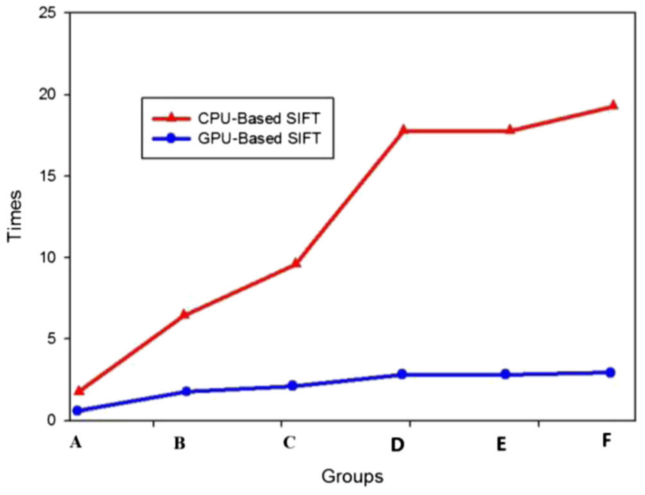

**Fig. 4** Time comparison curves of the SIFT algorithm on CPU and GPU

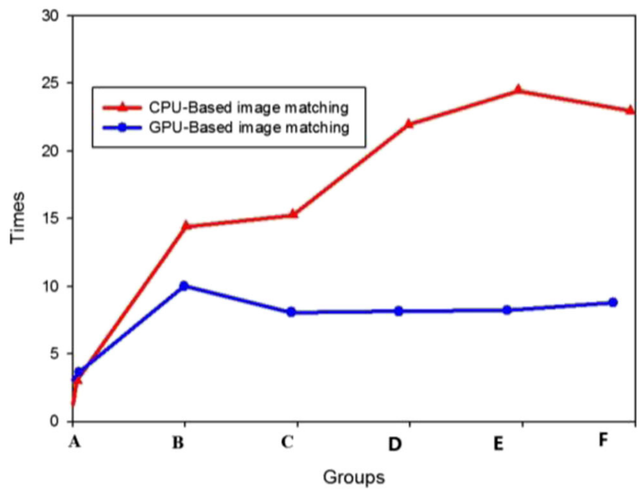

**Fig. 5** Time comparison of image matching on CPU and GPU

## 6 Conclusions

Scale-invariant feature transform (SIFT) is one of the widely used interest point features. It has been successfully applied in various computer vision algorithms like object detection, object tracking, robotic mapping, and large-scale image retrieval. Although SIFT descriptor is highly robust towards scale and rotation variations, the high computational complexity of the SIFT algorithm inhibits its use in applications demanding real-time response and in algorithms dealing with very large-scale databases. In order to be effective for image matching process in near real-time, the Compute Unified Device Architecture (CUDA) application programming interface of a graphics processing unit (GPU) is cooperated to speed up or improved SIFT method. Experimental results show that the proposed GPU-based SIFT framework is suitable for image application with real-time. It can improve the image matching process both in time and accuracy compared with conventional SIFT method. In the future, we will apply our proposed GPU framework to UAV system so as to achieve high-precision wide-area observation.

## References

1. Li Z, Jia H, Zhang Y (2017). HartSift: A high-accuracy and real-time SIFT based on GPU[C]. IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS). IEEE Computer Society.
2. Lalonde M, Byrns D, Gagnon L et al (2007) Real-time eye blink detection with GPU-based SIFT tracking[C]. IEEE Fourth Canadian Conference on Computer and Robot Vision (CRV '07).
3. Warn S, Emeneker W, Cothren J, Apon A (2009) Accelerating SIFT on parallel architectures[C]. IEEE International Conference on Cluster Computing and Workshops
4. Fassold H, Rosner J (2015) A real-time GPU implementation of the SIFT algorithm for large-scale video analysis tasks[C]. Real-Time Image and Video Processing 2015. International Society for Optics and Photonics
5. Wang G, Rister B, Cavallaro JR (2013) Workload analysis and efficient OpenCL-based implementation of SIFT algorithm on a smartphone[C]. 2013 IEEE Global Conference on Signal and Information Processing
6. Acharya KA, Babu RV (2013) Speeding up SIFT using GPU[C]. IEEE Fourth National Conference on Computer Vision, Pattern Recognition, Image Processing and Graphics (NCVPRIPG)
7. Mohammadi MS, Rezaeian M (2014) Towards affordable computing: SiftCU a simple but elegant GPU-based implementation of SIFT[J]. Int J Comput Appl 90(7):30–37
8. Acharya KA, Venkatesh Babu R, Vadhiyar SS (2018) A real-time implementation of SIFT using GPU[J]. J Real-Time Image Process 14(2):267–277
9. Jiang C, Geng ZX, Wei XF, et al (2013) SIFT implementation based on GPU[C]. International Symposium on Photoelectronic Detection and Imaging 2013: Optical Storage and Display Technology. Int Soc Optics Photonics
10. Zhan ZF, Li G, Zhang XH (2014) Research on SIFT matching algorithm based on GPU[J]. Appl Mech Mater 599–601:1652–1656
11. Dadi EW, Daoudi EM (2014) GPU-based for accelerating the BF-SIFT method for large scale 3D shape retrieval[C]. IEEE International Conference on Multimedia Computing & Systems.
12. Lee C, Rhee CE, Lee H (2017) Complexity reduction by modified scale-space construction in SIFT generation optimized for a mobile GPU[J]. In IEEE Transactions on Circuits and Systems for Video Technology 27(10):2246–2259
13. Daðason K, Lejsek H, Jóhannsson ÁÞ et al (2010) GPU acceleration of Eff2 descriptors using CUDA[C]. ACM International Conference on Multimedia
14. Jiang J, Li X, Zhang G (2014) SIFT Hardware Implementation for Real-Time Image Feature Extraction[J]. In IEEE Transactions on Circuits and Systems for Video Technology 24(7):1209–1220
15. Kumar RP, Muknahallipatna SS, Mcinroy JE (2014) SIFTs scale-space extrema detection on GPU for real-time applications (WIP)[C]. Summer Simulation Multiconference. Soci Comp Simulation Int
16. Kusamura Y, Kozawa Y, Amagasa T, Kitagawa H (2016) GPU Acceleration of content-based image retrieval based on SIFT descriptors[C]. 19th International Conference on Network-Based Information Systems (NBiS), Ostrava, 2016, pp 342–347
17. Donglin J, Jianzhi L, Wanwan Y, Ye J (2015) Parallel realization of SIFT feature abstraction based on GPU using TEGRA K1[C]. IET International Radar Conference 2015, Hangzhou, 2015, pp 1–4
18. Cheng J, Zhu X, Ding W, Gao G (2016) A robust real-time indoor navigation technique based on GPU-accelerated feature matching[C]. 2016 International Conference on Indoor Positioning and Indoor Navigation (IPIN), Alcala de Henares, 2016, pp 1–4
19. Rister B, Wang G, Wu, Cavallaro JR (2013) A fast and efficient sift detector using the mobile GPU[C]. 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, 2013, pp. 2674–2678
20. Sun Y, Zhao L, Huang S, Yan L, Dissanayake G (2014) L2-SIFT: SIFT feature extraction and matching for large images in large-scale

aerial photogrammetry[J]. ISPRS Journal of Photogrammetry and Remote Sensing, volume 91, 2014, pp 1–16

21. Warn S, Apon A, Cothren J (2011) Accelerating SIFT on hybrid clusters. In Proceedings of the ACM SIGSPATIAL Second International Workshop on High Performance and Distributed Geographic Information Systems (HPDGIS ’11)
22. Bhangale U, Durbha S (2015) High performance SIFT features clustering of VHR satellite images for disaster management[C]. In Proceedings of the Third International Symposium on Women in Computing and Informatics (WCI ’15), Indu Nair (Ed.). ACM, New York, p 324–329