A parallel Canny edge detection algorithm based on OpenCL acceleration

Yupu Song
Shangqiu Polytechnic
Cailin Li
Shandong University of Technology
Qinglei Zhou
Zhengzhou University
Han Xiao (xiaohan70@163.com)
Zhengzhou Normal University

Research Article

Keywords: Canny algorithm, Edge detection, Graphics Processing Unit (GPU), Open Computing Language (OpenCL), Parallel algorithm

Posted Date: April 6th, 2023

DOI: https://doi.org/10.21203/rs.3.rs-2774366/v1

Additional Declarations: No competing interests reported.

A parallel Canny edge detection algorithm based on OpenCL acceleration

Yupu Song[1] · Cailin Li[2] · Qinglei Zhou[3] · Han Xiao[4]

Received: 4 April 2023 / Revised: 20 May 2023 / Accepted: 20 May 2023

Abstract

In the process of Canny edge detection, a large number of high complexity calculations such as Gaussian filtering, gradient calculation, non-maximum suppression, and double threshold judgment need to be performed on the image, which takes up a lot of operation time, which is a great challenge to the real-time requirements of the algorithm. In order to solve this problem, a fine-grained parallel Canny edge detection method is proposed, which is optimized from three aspects: task partition, vector memory access, and NDRange optimization, and CPU-GPU collaborative parallelism is realized. At the same time, the parallel Canny edge detection methods based on multi-core CPU and CUDA architecture are designed. The experimental results show that OpenCL accelerated Canny edge detection algorithm can achieve 20.68 times, 3.96 times, and 1.21 times speedup ratio compared with CPU serial algorithm, CPU multi-threaded parallel algorithm, and CUDA-based parallel algorithm, respectively. The effectiveness and performance portability of the proposed Canny edge detection parallel algorithm are verified, and it provides a reference for the research of fast calculation of image big data.

Key words Canny algorithm · Edge detection · Graphics Processing Unit (GPU) · Open Computing Language (OpenCL) · Parallel algorithm

1 Introduction

With the development of computer science, image processing technology has achieved fruitful research results in recent years and has been widely used in industrial, military, medical, and other fields. As the most basic feature of the image, the edge feature of the image can greatly reduce the image information to be processed on the premise of retaining the shape information of the object [1]. The edge of a digital image contains a variety of useful information, which can be used to detect and recognize images. Digital image edge detection technology is widely used in image segmentation, motion detection, target tracking, face recognition, and other fields. Therefore, edge detection is one of the most important key technologies in the field of image processing [2].

At present, image edge detection algorithms mainly include edge detection algorithms based on wavelet transform, edge detection algorithms based on morphology, edge detection algorithms based on machine learning, and traditional edge detection algorithms [3]. The edge detection algorithm based on wavelet transform is used to transform the image with different scales. When the scale is small, the edge detail information is rich, and the positioning accuracy is high, but the anti-disturbance ability is poor. When the scale is large, the positioning accuracy is low and the anti-jamming ability is good, so it fuses the results of edge images of each scale, taking into account the positioning accuracy and anti-jamming ability to a certain extent, but the algorithm complexity

✉ CaiLin Li
licailin@sdut.edu.cn

Han Xiao
xiaohan70@163.com

[1] College of Computer Engineering, Shangqiu Polytechnic, Shangqiu, China

[2] School of Civil and Architectural Engineering, Shandong University of Technology, Zibo, China

[3] School of Computer and Artificial Intelligence, Zhengzhou University, Zhengzhou, China

[4] School of Information Science and Technology, Zhengzhou Normal University, Zhengzhou, China

is high [4]. The edge detection algorithm based on morphology uses the continuous movement of structural elements in the image to analyze and process the image and extracts different image features by selecting different structural elements for opening and closing and other operations. This algorithm is easy to implement, and can effectively remove the salt and pepper noise, but its edge location accuracy is not good. Edge detection algorithm based on machine learning has become a new research direction in recent years. In particular, the deep features of the image are extracted automatically by deep learning, and a good edge effect is obtained. Its disadvantage is that it requires a large number of samples of training and learning, and the computational complexity is high [5].

The traditional edge detection algorithms include the Roberts operator, Prewitt operator, Sobel operator, and so on. These algorithms are simple and easy to implement, but their denoising ability is poor, crack edge recognition is incomplete, and pseudo edges are easy to occur. Compared with these algorithms, the Canny edge detection operator used in this paper has a strong denoising ability and high detection accuracy [6]. The Canny edge extraction method was first proposed by John F.Canny in 1986 [7]. The Canny edge detection method is based on finding the local maximum of the gradient amplitude of the image. It uses the first derivative of the Gaussian filter to calculate the gradient amplitude. It uses the double-domain value method to detect the strong and weak edges respectively, and only when the strong and weak edges are connected, the weak edges where the strong edges are discontinuous will be included in the detection results. As a result, the influence of noise on the detection results can be reduced, and the detection results can achieve a better balance between noise and edge detection. However, the Canny operator also has obvious shortcomings. Due to the calculation flow of Gaussian filtering, gradient amplitude and direction calculation, non-maximum suppression, and double threshold processing, the algorithm has high complexity and slow operation speed, which is contrary to the fast and accurate application principle in practical engineering, which greatly restricts the engineering practicability of the algorithm. In order to improve the computing speed of the Canny operator, it is a good choice to use Graphics Processing Unit (GPU) to parallelize processing. GPU has multiple threads for fast computing of large data with low coupling and high parallelism. At the same time, the parallel computing of GPU is becoming more and more mature in recent years, and its friendly programming operation and people-friendly price also make it possible to use GPU parallel processing Canny operator [8, 9]. In order to use GPU parallel processing Canny operator, it is necessary to optimize and parallelize the processing process of the Canny operator, so as to meet the requirements of GPU parallel processing. Through the optimization and transformation of the Canny operator, the processing mode of running GPU+CPU reduces the edge detection time of a 1280 × 720 image to less than 10 ms, which greatly improves the execution efficiency of the algorithm and lays a foundation for practical industrial applications.

For the problem that it is difficult to have both effectiveness and performance portability, this paper re-evaluates and analyzes all the steps of Canny edge detection according to the architecture of GPU, so that the key hot steps run completely on GPU. Based on the architecture of Open Computing Language (OpenCL), the parallel implementation of the Canny edge detection algorithm (OCL_Canny) is completed. By analyzing the conventional inefficient memory access mode of single work-item and single pixel and the deficiency of low utilization of GPU memory, the method of vectorized memory access is proposed, which improves resource utilization and computational efficiency. At the same time, the OCL_Canny parallel algorithm also has the advantages of real-time and performance portability.

Therefore, the main contributions of this paper are as follows: (1) implement the Canny edge detection algorithm OCL_Canny through heterogeneous computing. (2) The OMP_Canny and CUDA_Canny parallel algorithms under the mainstream parallel computing framework of OpenMP and Compute Unified Device Architecture (CUDA) compare the time-consuming and accelerated performance with the OCL_Canny algorithm. (3) The performance of OCL_Canny on a heterogeneous GPU platform is evaluated, and the portability of its performance is analyzed.

The rest of the paper is arranged as follows. In Section 2, we review the research results of the Canny edge detection parallel algorithm, the existing implementation of FPGA and DSP computing architecture, the existing computing methods on graphics hardware, and the Canny algorithm on Hadoop cluster system. Section 3 summarizes the basic principles of OpenCL architecture and describes the Canny edge detection

algorithm and the parallelism analysis of Canny operators. Section 4 describes the parallel computing process, design, and optimization solution of the Canny operator under OpenCL architecture. Section 5 discusses the design of OMP_Canny and CUDA_Canny parallel algorithms. Section 6 gives the relevant experimental results and makes an empirical evaluation of the performance of the OCL_Canny operator. Section 7 is the conclusion.

2 Background and introduction of related research

At present, many researchers have researched the implementation of the Canny edge detection parallel algorithm. SHI Weizhong et al. [10] proposed an optimization algorithm of Canny edge detection based on FPGA, which is suitable for real-time processing in deep space optical autonomous navigation. Jin et al. [11] chose ZC706 as the development platform to accelerate the edge detection of Canny based on the SDSoC development environment and achieved a speedup of 16.97 times. Keqiang et al. [12] developed a Canny operator on the TI DSP TMS320C6678 processor, which improves the speed of the operator. Xiangjiao et al. [13] implemented a parallel Canny algorithm based on Threading Building Block (TBB) tool and C++ language and achieved 3.673 times acceleration ratio on a quad-core CPU. Yue et al. [14] realized the Canny edge detection algorithm on GPU using OpenGL, and the real-time performance of the algorithm was satisfied. Bin et al. [15] proposed a method to quickly implement the Canny operator based on GPU+CPU, with a speedup of up to 5.39 times. Jin et al. [16] proposed a Canny edge detection algorithm under OpenCL architecture, which achieves 6.16 times speedup without considering data transmission. Some scholars have studied the implementation of the Canny edge detection algorithm in Hadoop cluster architecture, which improved the performance of batch processing images [17, 18].

Some scholars have proposed an improved Canny image edge detection method, which can effectively detect the image edge in real time on FPGA [19, 20]. Lee et al. [21] implemented a Canny edge detector suitable for advanced mobile vision applications on FPGA under the slight sacrifice of detection effect, which saves the execution time of the system. Suwen et al. [22] proposed an improved Canny edge detection algorithm based on the FPGA platform, which improves the ability of weak edge detection. Shengxiao et al. [23] proposed an improved algorithm for edge detection of the Canny operator based on the GPU platform, which obtains 64 times speedup.

Fuqiang et al. [24] designed the line segment detector algorithm with low error rate by using the Canny edge detection algorithm implemented on FPGA, which has the advantages of high reliability and high speed. Sivakumar et al. [25] proposed a new ROI region segmentation method for MRI images by implementing enhanced Canny operators on FPGA. Hongye [26] realized the fingerprint acquisition system based on DSP by optimizing the Canny edge extraction operator, which makes the identification speed of the fingerprint wireless acquisition system faster. Rongbao et al. [27] designed a verticality recognition system based on DSP+FPGA using the improved Canny algorithm. The results show that the system has high detection speed, and high precision and meets real-time requirements. Hanjun et al. [28] combined the Gaussian mixture model with Canny edge detection to extract the target contour, which shortens the computing time on the CUDA platform and meets the real-time requirements of video analysis. Tengzhang et al. [29] proposed a method based on the multi-feature Canny edge detection algorithm and the joint probability data association algorithm for moving multi-ship detection and tracking by on-orbit satellite. This method can detect and track the target quickly and accurately on the embedded GPU development platform.

To sum up, people mainly study the performance of the Canny algorithm from three aspects. The first is to accelerate the Canny operator in parallel under the architecture of FPGA, DSP, GPU, and Hadoop clusters. Although these research results have achieved a certain degree of performance improvement, the speedup is not high, the computing time is not ideal. The second is to improve the performance of the improved Canny operator by improving the Canny operator in some aspects, such as optimizing the calculation process. The third is to apply the Canny operator to a variety of practical applications to achieve the acceleration of the application system under the parallel computing architecture. However, on the one hand, the acceleration effect of these research results is not ideal. On the other hand, Canny edge detection often uses a single parallel technology to improve the algorithm, without comparison with other parallel computing models, it cannot get the best acceleration effect. Current computer systems generally contain a variety of processors, such as CPU, GPU, and other types of processors. How to make reasonable and

full use of a variety of computing resources on heterogeneous computing platforms will become very important.

In this paper, the storage of GPU is designed and used reasonably by using OpenCL parallel acceleration technology to realize the high-speed computing of the Canny image edge detection algorithm. By taking the three memory access modes of image data access on GPU, namely, global memory, local memory, and constant memory, as a starting point, the parallel implementation of image Gaussian filtering and image gradient in these three kinds of memory is analyzed and designed, so that the two operations can be realized more efficiently on GPU. In the process of research, GPU is used to realize image Gaussian filtering, image gradient, non-maximum value suppression of gradient image, and determining image edge points in parallel. To obtain the fast extraction of image edge as the goal, the calculation methods of image Gaussian filtering and image gradient are optimized and improved. From the perspective of saving storage resources and being more in line with the parallel programming architecture of GPU, the computing is improved, such as improving the operation method of the image template and extended image to make it more suitable for the parallel implementation under GPU. At the same time, the construction of pixel vectorization calculation under GPU is applied to the calculation of image Gaussian filtering and image gradient, which verifies the effectiveness of the vectorization parallel computing method of image Gaussian filtering and image gradient calculation.

3 Software model of the Canny algorithm

3.1 Overview of OpenCL

OpenCL is used for a parallel computing platform, which establishes the writing standard of parallel systems. OpenCL has a relatively wide range of applications, providing computing support for CPU, GPU, FPGA, and other devices, and has become a programming standard in the field of heterogeneous systems. OpenCL provides developers with a common programming interface and a development model for the underlying hardware layout [30].

OpenCL heterogeneous parallel architecture consists of four parts: platform model, execution model, storage model, and programming model. The four models support each other when the OpenCL system is running, and each model has its own unique role.

(1) Platform model

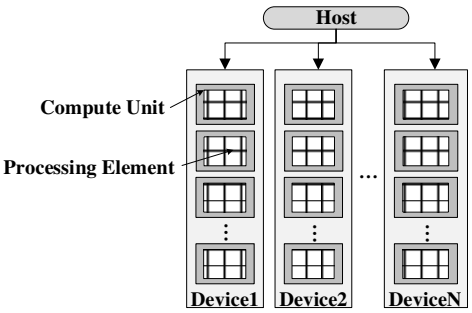

Fig.1 OpenCL platform model

As shown in Figure 1, the OpenCL platform model consists of a Host connected to one or more OpenCL compute devices, which is used to realize the data exchange between the host and the OpenCL devices. CPU, GPU, and other processors that support OpenCL all belong to OpenCL devices. An OpenCL device can be divided into one or more Compute Units (CU), and each CU is composed of one or more Processing Elements (PE) [31].

(2) Memory model

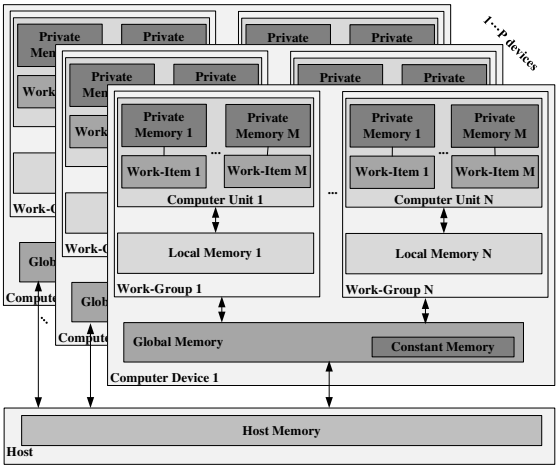

Fig.2 OpenCL memory model

The memory in OpenCL architecture is divided into four different memory types. The location of each memory in the platform is shown in Figure 2. These four types of memory are global memory, constant memory, local memory, and private memory [32].

(3) Execution model

The execution model is shown in Figure 3. The execution model of OpenCL consists of two parts, one is the host system executing on the host machine, and the other is the kernel software executing on the OpenCL device. The OpenCL architecture manages the execution of kernel software in OpenCL devices by using context in the main system [33].

When the send kernel command is submitted on the host, the system plans an N-dimensional index space NDRang. The operation of each point in this space is called a work, which OpenCL calls a work-item. All work-items in the index space have their own unique coordinates, which serve as the global ID for each work-item. When sending kernel execution commands, the work-item is divided into several areas of the same size and becomes a collection of work-items, which are called work-groups. The number of work-items contained in all work-groups is the same, and similar to the global ID of work-items, work-groups also have ID, called work-group ID. Work-items in each work-group have a unique ID in the work-group, called a local ID. Figure 3 gives a two-dimensional index space, the size of the index space is $Gx * Gy$, in which a coordinate system is established to represent the global ID (gx, gy) of each work-item. The index space in the graph is divided into multiple work-groups with $Sx * Sy$ work-items. OpenCL stipulates that Gx must be divisible by Sx and Gy must also be divisible by Sy [34].

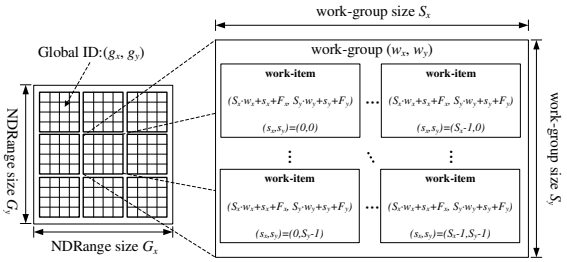

Fig.3 OpenCL execution model

(4) Programming model

OpenCL achieves the goal of acceleration by executing tasks in parallel, which is divided into task parallelism and data parallelism. Task parallel mode means that all the working nodes in the workspace of OpenCL devices are relatively independent, and the system can accelerate by executing multiple kernels at the same time or adding local kernel tasks to the kernel. Data parallel methods are commonly used, and multiple data are calculated in parallel so that the computational efficiency is significantly improved.

3.2 Algorithm theory

3.2.1 Canny principle

The Canny operator fully reflects the mathematical characteristics of the optimal edge detector. It is the optimal approximation operator for the signal-to-noise ratio and location ability and is widely used in image processing and pattern recognition problems. The Canny operator not only has a good edge detection performance but also is insensitive to noise, even in a noisy environment, it also has a good edge detection effect. Therefore, the Canny operator can be applied to edge detection in different environments.

(1) Image preprocessing

The images to be detected are usually disturbed by noise. The amplitude of the gradient near the noise pixel is large, and the edge detection operator is easy to mistakenly detect the noise pixel as the edge pixel. Therefore, it is necessary to remove the noise in the image.

When the image is used for edge detection, the original data must be processed first. The input image is preprocessed by convolution filter with Gaussian filter to remove noise and reduce the influence of noise on gradient calculation, so as to better realize the effect of edge detection image segmentation. Therefore, image preprocessing requires convolution of the original image and Gaussian mask, and the processed image is more blurred than the original, which is conducive to image edge detection [35].

In the Canny operator, the smooth denoising of the image uses the first derivative of the 3 × 3 two-dimensional Gaussian function, and the Gaussian function and image convolution are shown in equation (1).

$$\begin{cases} G(x, y, \sigma) = \dfrac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \\ H(x, y) = f(x, y) * G(x, y, \sigma) \end{cases} \tag{1}$$

In equation (1), $f(x, y)$ is the original image, $G(x, y, \sigma)$ is the Gaussian function, σ is the standard deviation of the two-dimensional Gaussian function, and $H(x, y)$ is the image smoothed by the Gaussian filter.

(2) Determine the amplitude and direction of the image gradient

The amplitude of the pixel gradient of the image $H(x, y)$ can be calculated by the first partial derivative. In calculating the gradient direction, two 3 × 3 Sobel operators are used as the first order approximation of the partial derivatives in the x direction and y direction, as shown in Figure 4 [36].

-1	-2	-1	-1	0	1
0	0	0	-2	0	2
1	2	1	-1	0	1

Fig.4 Sobel operator template

Before determining the amplitude and direction of the image gradient, equation (2) is used to solve the first order partial derivative matrix of the x -axis and y -axis direction.

$$\begin{cases} P(x, y) = H(x+1, y-1) + 2H(x+1, y) + H(x+1, y+1) \\ \quad - H(x-1, y-1) - 2H(x-1, y) - H(x-1, y+1) \\ Q(x, y) = H(x-1, y+1) + 2H(x, y+1) + H(x+1, y+1) \\ \quad - H(x-1, y-1) - 2H(x, y-1) - H(x+1, y-1) \end{cases} \tag{2}$$

The amplitude and direction of the gradient are calculated by the finite difference of the first order partial derivative. For the calculation results of the gradient amplitude, the non-maximum value suppression method is adopted. After processing, the gradient amplitude M and gradient direction θ at the pixel $H(x, y)$ of the image can be calculated by equation (3) and equation (4) respectively [37].

$$M(x, y) = \sqrt{P(x, y)^2 + Q(x, y)^2} \tag{3}$$

$$\theta(x, y) = \arctan\left[\frac{Q(x, y)}{P(x, y)}\right] \tag{4}$$

(3) Perform non-maximum value suppression on the gradient amplitude image to determine the edge point

Non-maximum value suppression is the key to find all the target edge points in the image. In order to determine the edge, it is necessary not only to get the global gradient but also to retain the maximum point of the local gradient and suppress the non-maximum value. In the 3 × 3 region, the edge can be divided into four directions: 0°, 45°, 90° and 135°. Similarly, the reverse direction of the gradient is also four directions (orthogonal to the edge direction). Therefore, in order to suppress the non-maximum value, all possible directions are quantized into four directions, as shown in Figure 5 [38].

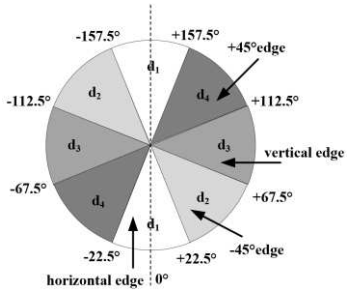

Fig.5 Sector chart

In this way, the direction angle is regulated to the following four directions:

The vertical edge — gradient direction is horizontal: $\theta(x, y) \in [67.5°, 112.5°] \cup [-112.5°, -67.5°]$

The 135° edge — gradient direction is 45°: $\theta(x, y) \in [22.5°, 67.5°) \cup [-157.5°, -112.5°)$

The horizontal edge — gradient direction is vertical: $\theta(x, y) \in [0°, 22.5°) \cup (-22.5°, 0°] \cup (157.5°, 180°] \cup (-180°, -157.5°]$

The 45° edge — gradient direction is 135°: $\theta(x, y) \in (112.5°, 157.5°] \cup [-67.5°, -22.5°]$

In the 3 × 3 region, for each pixel in the image, there are only four possible directions connected to the adjacent points: 0°, 45°, 90°, and 135°, as shown in Figure 6 [39].

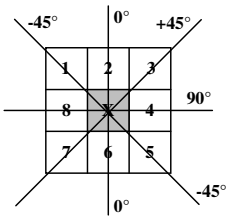

Fig.6 Pixel neighborhood structure

At the central pixel $H(x, y)$ of each neighborhood is compared with two adjacent pixels along its corresponding gradient direction $\theta(x, y)$. If the gradient value $M(x, y)$ at the center point is the largest, then the corresponding $f(x, y)$ grayscale value is retained, otherwise, $f(x, y)$ grayscale value is set to 0. As a result, the non-maximum value suppressed image $f'(x, y)$ is obtained.

(4) Using double threshold algorithm to detect and connect edges of gradient images

In order to reduce the pseudo edge points, the double threshold algorithm is used to distinguish and connect the edges. If the edge strength is greater than the high threshold, it must be the edge point. If the edge strength is less than the low threshold, it must not be the edge point. If the edge intensity is greater than the low threshold and less than the high threshold, then see if there are any edge points in the adjacent pixels of this pixel that exceed the high threshold, if so, it is the edge point, if not, it is not the edge point [40].

Two thresholds, T_L and T_H, are selected with a ratio of 1:2 or 1:3. For the image $f'(x, y)$ processed by non-maximum value suppression processing, if the gradient value of the pixel is $M(x, y) \geqslant T_H$, then the pixel is marked as an edge pixel, namely, and the $f(x, y)$ grayscale value is set to 255. If the gradient value of the pixel is $M(x, y)$,, T_L, then the pixel is marked as a non-edge pixel, namely, and the $f(x, y)$ grayscale value is set to 0. If the gradient value of the pixel is $T_L < M(x, y) < T_H$, then the pixel is marked as "quasi-pixel", that is, and the $f(x, y)$ grayscale value is set to 1. After the double threshold marking is completed, search for "quasi-pixel points" in the image, and select the positions of its 8 neighborhood points to find out whether there is a point with gradient value $M(i, j) \geqslant T_H$. If it exists, mark the pixel as an edge point, otherwise mark the pixel as a non-edge pixel.

3.2.2 Eliminate branches

When using a template to traverse an image, the computation is out of bounds when traversing to the edge of the image. Therefore, the edge of the image to be processed is expanded before the calculation begins. The method of dealing with edge pixels in this paper is to make full use of the similarity of the image and take its own pixels to expand the original image. Suppose that the size of the original image is $H \times H$, and the size of the image after edge expansion is $H' \times H'$, as shown in Figure 7, the solid line region and the dotted line region, respectively. When the neighborhood size is $n \times n$, the edges of $\lfloor n/2 \rfloor$ pixels are filled around the original image.

After extended preprocessing, there is no need for branch processing, which ensures a high degree of unity of the implementation process, and then improves the parallel potential of the algorithm.

In this paper, the method of even expansion is used to expand the edge of the original image. First of all, the gray values of all the pixels of the original image are filled into the middle part of the expanded edge image in turn. Then, fill the left boundary data of the original image into the corresponding left expansion area of the flared image, as pointed by the red arrow in Figure 7. Fill the right boundary data of the original image into the corresponding right expansion area of the flanged image, as pointed by the green arrow in Figure 7. Finally, fill the upper boundary data of the expanded image into the corresponding upper expansion area (including corners) of the final edge image, as pointed by the black arrow in Figure 7, and fill the lower boundary data of the expanded image into the corresponding lower expansion area (including corners) of the final edge image, as pointed by the blue arrow in Figure 7.

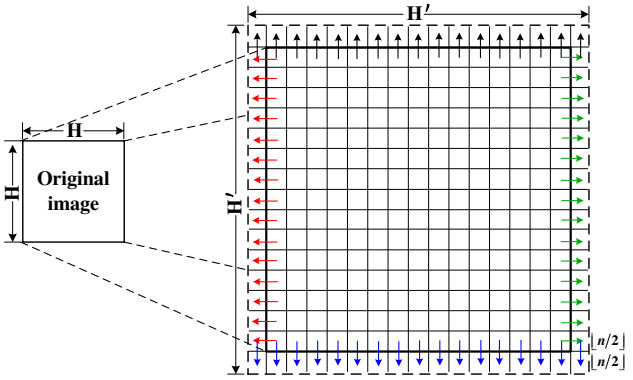

Fig.7 Boundary processing

3.3 Serial system analysis

The 1024 × 1024 image size was used to test, the data bit depth was 8 bits, and the data format was BMP. When the CPU is Intel Core i7-8700K and the filter neighborhood size is 3 × 3, the time-consuming of each calculation step on the CPU is shown in Table 1. It can be seen from Table 1 that the most time-consuming step of the whole algorithm is the calculation of Canny edge detection, which includes the Gaussian filtering process for noisy images. The Canny edge detection step accounts for about 79.72% of the processing time of

the whole Canny system. Therefore, the parallel acceleration in this paper will mainly focus on the Canny edge detection part.

Table 1 Time-consuming of each module in the Canny algorithm

Algorithm steps	Time-consuming by CPU (ms)	Occupancy time ratio (%)
Read in source image data	2.23	1.20
Extended source image	3.65	1.97
Gaussian template calculation	10.24	5.53
Initialize non-maximum value suppressed image	19.45	10.50
Canny edge detection	147.69	79.72
Output image edge extraction result	2.01	1.08
Total	185.27	100.00

In the calculation process of Canny edge detection, firstly, Gaussian filtering needs to take a filter window around the calculation point, and convolution calculation is carried out in this window. Then, the amplitude and direction of the image gradient need to be determined by using the Sobel operator, and then the gradient amplitude image is suppressed by non-maximum value, thus the non-maximum value suppression image is obtained. Finally, the double threshold algorithm is used to distinguish and connect the edges. Each pixel in the image data is processed in turn. When the image scale is large, the system will produce a large amount of computation. Therefore, reducing the computing time of Canny edge detection processing is one of the problems to be solved in this algorithm.

Suppose, the image size is $H \times H$ and the neighborhood size is $n \times n$. Then

Process 1: The time complexity of the process of initializing a non-maximum value suppression image is $O\left(H^2\right)$.

Process 2: The time complexity of the step of expanding the edge of the image is $O\left(H^2\right) + O\left(H \times n\right)$.

Process 3: The time complexity of the image Gaussian filtering step is $O\left(H^2 n^2\right)$.

Process 4: The time complexity of the process of determining the amplitude and direction of the image gradient is $O\left(H^2 n^2\right)$.

Process 5: The time complexity of non-maximum value suppression of gradient amplitude image is $O\left(H^2\right)$.

Process 6: The time complexity of the process of detecting and connecting edges of gradient images is $O\left(9H^2\right)$.

Therefore, the total time complexity of the Canny edge detection algorithm is: $2O\left(H^2 n^2\right) + 3O\left(H^2\right) + O\left(H \times n\right) + O\left(9H^2\right)$. From the above analysis, it can be seen that process 3 ~ 6 is a functional part of the Canny edge detection algorithm with relatively high time complexity. Therefore, this paper should mainly focus on the parallel optimization of process 3 ~ 6, that is, the stage of Canny edge extraction. To sum up, the time complexity of the Canny edge detection algorithm is $O\left(H^2 n^2\right)$.

3.4 Algorithm parallel analysis

The parallelism analysis of the hot step process 3—process 6 in the Canny edge detection algorithm is carried out, and the time complexity of the algorithm is analyzed.

(1) Process 3: From the point of view of the image Gaussian filtering process, the $n \times n$ point multiplication is mainly carried out through the image pixel matrix and the Gaussian template matrix. The bottom layer of the algorithm processes a large amount of data, but the operation process is relatively simple. All pixels in the image can perform the same operation, there is no data dependence between each point of the target matrix, these operations can be performed in parallel, and the algorithm is a memory-intensive algorithm. In view of this, this paper realizes the optimization of the algorithm by improving the memory access efficiency and making rational use of GPU hardware resources.

(2) Process 4: The calculation of the amplitude and direction of the image gradient is to convolution each pixel with the Sobel operator in the *x* direction and *y* direction respectively, and then calculate the amplitude and direction of the gradient for the pixel. These computing processes are independent of each other and can be calculated in parallel.

(3) Process 5: Each central pixel is compared with two adjacent pixels in the same gradient direction to suppress non-maximum value pixels. The comparison process of each

group is only related to the amplitude data of the current comparison pixels, but has nothing to do with other pixels. Each group of comparison processes can correspond to a work-item, so that process 5 can be executed in parallel.

(4) Process 6: The process of judging the edge points of pixels by using double thresholds does not affect each other and is independent of each other. It is beneficial to give full play to the performance advantages of GPU devices.

To sum up, the hot steps of the Canny edge detection algorithm, process 3 — process 6, can be executed in parallel, which is suitable for implementation on GPU. Therefore, a work-item is created for each pixel so that the corresponding pixel can be processed accordingly. Because all work-items perform the same computing process at the same time, the time complexity of the Canny edge detection parallel algorithm will be reduced to $O(n^2)$, which is a very small level of complexity. If all pixels are not processed in one kernel function, each work-item will perform the Canny edge detection kernel function at least $H^2/tsum$ times, where *tsum* is the number of work-items. In this case, the time complexity of the Canny edge detection parallel algorithm will be $O(H^2n^2/tsum)$. It is important to note that because of the large number of active work-items that can be maintained in GPU, *tsum* is always a large value. Therefore, there exists the time complexity of the Canny parallel algorithm $O(H^2n^2/tsum) = O(H^2n^2)$.

4 OpenCL implementation of Canny edge detection algorithm

4.1 Parallel algorithm description

In order to maximize the effective use of GPU hardware multi-work-item resources, the reconstruction algorithm must strictly follow the OpenCL multi-work-item framework processing concept. In the process of image Gaussian blur, amplitude and direction calculation of image gradient, non-maximum value pixel suppression, and edge point judgment by GPU, the important foundation is that there is no correlation between pixel-by-pixel calculation. That is, the processing of each pixel is not related to each other. According to this characteristic, the Canny edge detection task can be divided into several different kernels using GPU, and the image pixels can be processed and calculated in parallel by multiple work-items in the kernel. The specific Canny edge extraction parallel algorithm is shown below.

1: **Algorithm1 Canny edge detection parallel algorithm on OpenCL**
2: **Input**: Noisy image matrix *srcImageData* with image size $H \times H$, array $GaussTemplate[0:n\times n-1]$ of the Gaussian convolution kernel, the array $SobelTemplate[0:n\times n-1]$ of the Sobel convolution kernel, each work-item is responsible for Gaussian filtering and processing of Sobel convolution in two directions of $BX \times BY$ pixels.
3: **Output**: Image matrix *desImageData* with canny edge detection
4: **Begin**
5: *CPU main function:*
6: *srcImageData* ← input image with an image size $H \times H$
7: *srcImageDataEx* ← extended original image
8: $GaussTemplate[0:n\times n-1]$ ← calculate the Gaussian filter template
9: *GPU kernel function:*
10: Initialize the global index *gx*, *gy* of the work-item in the *x* and *y* directions, respectively
11: Initialize the local index *lx*, *ly* of the work-item in the *x* and *y* directions, respectively
12: /* Gaussian filtering */
13: **for all** work-groups in NDRange **par-do**
14: Load the input sub-image data that a work-group need to access from the global memory into a
15: local memory of size *SubImage_ds*
16: **end for**
17: **for all** work-items in work-group **par-do**
18: **for** $i = 0$ **to** BX **do**
19: **for** $j = 0$ **to** BY **do**
20: **for** $f_x = 0$ **to** $n-1$ **do**
21: **for** $f_y = 0$ **to** $n-1$ **do**
22: $gaussPixel[i + j * BX]$ ←

Each work-item in the work-group does the

23: convolution operation result of the corresponding pixel and the Gaussian template

24: **end for**

25: **end for**

26: Output $gaussPixel[i+j*BX]$

27: **end for**

28: **end for**

29: **end for**

30: **/*** Amplitude and direction of image gradient ***/**

31: **for all** work-groups in NDRange **par-do**

32: Load the Gaussian filtering sub-image data that a work-group need to access from the global

33: memory into a local memory of size *SubImage_ds*

34: **end for**

35: **for all** work-items in work-group **par-do**

36: **for** $i=0$ **to** BX **do**

37: **for** $j=0$ **to** BY **do**

38: **for** $f_x=0$ **to** $n-1$ **do**

39: **for** $f_y=0$ **to** $n-1$ **do**

40: $convolution[i+j*BX] \leftarrow$

Each work-item in the work-group does the

41: convolution operation result of the corresponding Gaussian filtering image pixel

42: and the Sobel template

43: **end for**

44: **end for**

45: Calculate the gradient amplitude and direction of pixels

46: **end for**

47: **end for**

48: **end for**

49: **/*** Determine non-maximum suppressed image ***/**

50: **for all** work-items in NDRange **par-do**

51: Judge whether the gradient amplitude of the neighborhood center pixel is the largest in the gradient direction

52: **end for**

53: **/***Determine the edge points of the gradient image***/**

54: **for all** work-items in NDRange **par-do**

55: Using double threshold to judge whether the pixel of the gradient image is an edge point or not

56: **end for**

57: Transfer Canny edge detection results *desImageData* from global memory to host memory

58: **End**

4.2 Calculation process

The edge detection process of the OCL_Canny parallel algorithm is shown in Figure 8.

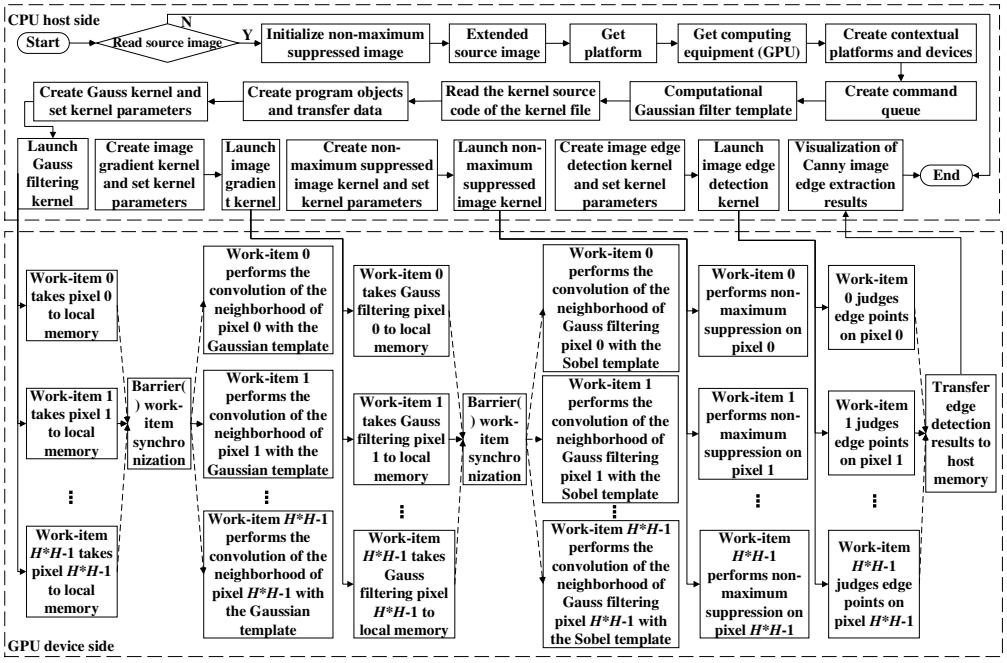

Fig. 8 OCL_Canny algorithm flow

The first step of the OCL_Canny parallel algorithm is to read the original image file to obtain image information and to expand the original image according to the size of the neighborhood window. Initialize the edge point image for subsequent calculation. Next, determine the platform for OpenCL execution, and then determine the device that performs the OpenCL calculation after determining the platform for execution. Create a context after determining the device.

After creating the context, you need to create a command queue. The operations such as extending the original image data transmission, Gaussian template data transmission, initializing the edge point image, and executing the kernel between the host and OpenCL devices are all done by queuing up to the command queue, and then the command queue passes each command to the OpenCL hardware unit for execution.

After that, the kernel code is compiled. First of all, the kernel source code is obtained from the host side and the program object is created, then the OpenCL device compiles and constructs the program object using the kernel source code, and finally constructs the kernel object to complete the compilation of the kernel code.

When the kernel function needs input parameters to provide calculation data, the corresponding application program interface function is called on the host side to complete the initialization of the input parameters. In addition, the work-group and work-item parameters used for execution on the device also need to be set in advance.

After the above operations are completed, the queuing operation is carried out, and the kernel function is sent to the corresponding command queue through the queuing command. The computing device interacts with the command queue and executes the corresponding kernel functions. The kernel functions of the OCL_Canny parallel algorithm include generating Gaussian filtered image kernel, generating gradient image kernel, generating edge point image kernel, and generating edge image kernel.

The operation of the kernel function is mainly the calculation and update of the incoming parameter variable, and the next call is the update status of the variable, and the four kernels are executed serially through CPU control. The execution process of the corresponding kernel function in this paper is as follows:

① Gaussian filtered image kernel. According to equation (1), the extended image data is convoluted with the Gaussian template data and the information is updated.

② Gradient image kernel. According to equation (2) ~ (4), the Gaussian smoothing image data is convoluted with the Sobel template data, and the gradient amplitude and direction of the corresponding pixels are calculated.

③ Edge point image kernel. The gradient image is suppressed by non-maximum value, and the edge points of the image are preliminarily determined.

④ Edge image kernel. The edge points of the image are finally determined and connected by the double threshold method.

After the OpenCL device performs the calculation, it transmits the results of Canny edge detection back to the host side and destroys the allocated resources.

4.3 Acceleration strategy of the algorithm

The Canny edge detection algorithm has obvious data computing parallelism. The processing of Gaussian filtering, calculating the gradient of the image, suppressing non-maximum value pixels, and judging edge points with double thresholds are only related to the position of the image pixels, and the calculation process of each pixel is exactly the same.

The mapping between the pixel and the OpenCL core mainly lies in the one-to-one logical correspondence between the work-item and the pixel. Figure 9 shows the mapping relationship between the NDRange workspace of the GPU and the image data matrix. The image frame $H \times H$ image data is arranged according to the one-dimensional linear organization in the system and can be decomposed into several non-overlapping sub-image blocks. Each sub-image block contains some pixels of the image. The kernel function creates

an NDRange workspace that identifies the index, as shown in the lower dotted frame in Figure 9. Through the mapping of OpenCL work-items to image pixels, each OpenCL work-item uses a unique work-item index to calculate the data that needs to be processed to achieve maximum parallelism.

Processing more data in a shorter time has always been one of the goals of high-performance computing. OCL_Canny parallel algorithm proposes a vectorization method to process multiple pixels at a time for each work-item. In the algorithm, the Gaussian filtering operation and the Sobel image gradient operation of four adjacent pixels in the sub-image block are scheduled on one work-item in turn. The calculation of the four output results is completed on the same work-item, and each cycle can complete the calculation of the output result of one pixel, thus completing the traversal of the four pixels. The coordinate transformation of the pixel is shown in equation (5).

$$\begin{aligned} lx &= get\_local\_id(0), \quad ly = get\_local\_id(1) \\ gx &= get\_global\_id(0), \quad gy = get\_global\_id(1) \end{aligned} \tag{5}$$

Among them, lx, ly represents the local ID of the work-item in the x, y direction respectively in the work-group. gx, gy represents the global ID of work-items in the x, y direction respectively in the workspace. Through the four variables, the precise scheduling of OpenCL work-items can be completed.

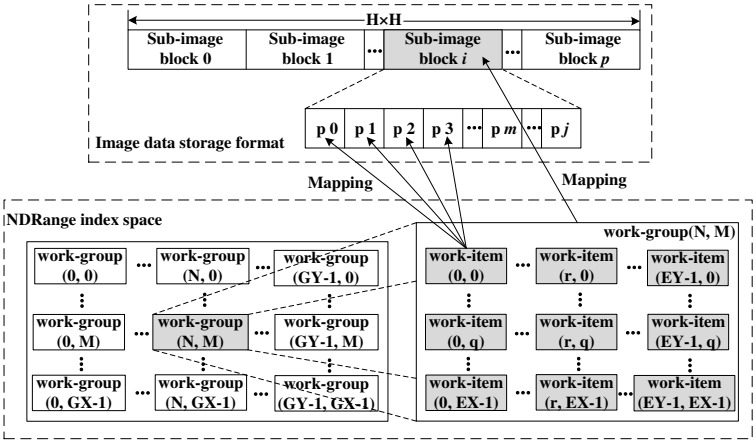

Fig. 9 Corresponding relation of the work-item index and image pixel coordinate

4.4 Algorithm optimization

4.4.1 Data storage adjustment

(1) Local memory optimization

In the processing of the four tasks of the Canny operator, the calculation of the boundary points in the work-group needs to cross the boundary. In order to prevent the image from crossing the boundary, the original image is extended to an expanded image in this paper. Suppose the template size is $n\times n$ and the original image size is $H\times H$. When calculating the Gaussian filter, gradient amplitude, and direction, the size of the expanded image is $(H+n-1)\times(H+n-1)$. In this case, the basic data area size of non-maximum value suppression and double threshold judgment of gradient image is $H\times H$, then the size of the extended data area is also $(H+n-1)\times(H+n-1)$.

It takes about 400 ~ 600 clock cycles to access data from global memory in the GPU, while it takes only 1~16 clock cycles to access data directly from local memory. The access speed of local memory is much higher than that of global memory. In the OCL_Canny parallel algorithm with global memory, it is necessary to access the global memory $(H+n-1)^2\times n^2\times 4$ times. By fetching the extended data area data from the global memory to the local memory, and taking the local memory as the memory for accessing the data when the work-item is calculated, the number of visits to the global memory is reduced to $(H+n-1)^2\times 2n\times 4$ times. Therefore, the optimized OCL_Canny parallel algorithm can significantly reduce the number of times of accessing global memory and greatly improve the access efficiency of the GPU.

(2) Constant memory optimization

In the OCL_Canny parallel algorithm, image preprocessing and image gradient calculation are completed under convolution computation. Since the convolution operation of the image needs to traverse the image pixels, when processing each pixel, it is necessary to read the corresponding pixels in the n^2 neighborhood to multiply and add with the Gaussian template and the Sobel template. It requires frequent access to memory and the calculation is very time-consuming. Considering that the constant memory has a cache mechanism when the access is hit, there is only one clock cycle delay. Therefore, in order to improve the access efficiency, the Gaussian template and the Sobel template are stored in the constant memory for all work-items to read. The constant memory has a 64 KB cache, and the storage space needed to store the Gaussian template and the Sobel template is $\left(3n^2 \times 4\right)$ B, which meets the maximum space requirements of the constant memory of 64 KB.

(3) Data reusability

From the processing flow of the Canny operator, we can see that each step of the algorithm is designed with a kernel, which is implemented with four kernels. Because there is a logical correlation between the kernel, that is, the results after the execution of the previous task need to be provided to the next task. Therefore, each kernel can store the calculation results in the global memory and wait for the next kernel to be read. By improving the data reusability, the data transmission times between CPU and GPU are reduced, thus the memory communication delay is hidden.

4.4.2 NDRange optimization

According to different GPU hardware, change the number of work-items in each work-group in the kernel function to achieve optimal performance. If the number of work-items is too small, it will cause most of the PEs to be idle, waste resources, and low performance. If the number of work-items is too large, due to the limitation of hardware resources, it may not be possible to actually start enough active work-items, which will cause too many work-items to be in a blocked state and also cause performance degradation. Therefore, in order to ensure the optimal performance of the OCL_Canny parallel algorithm on the GeForce GTX 1050 graphics card, the operation time of the algorithm is measured under different work-group dimensions, and the specific test data are shown in Table 2.

Table 2 Operation time of the OCL_Canny parallel algorithm

Image size	Parallel time corresponding to different work-group sizes (ms)				
	4×4	8×8	16×16	24×24	32×32
256×256	3.87	3.62	3.02	3.58	4.27
512×512	6.75	5.09	4.28	5.01	5.39
1024×1024	20.78	12.78	10.31	12.66	13.42

It can be seen from the above experimental results that for GeForce GTX 1050 graphics cards, the maximum number of work-items per work-group is 1024. An error will be reported when running over this number. At the same time, 16 × 16 is also the best operating efficiency point.

5 Other parallel schemes

5.1 The OMP_Canny parallel algorithm

The parallel processing of the Canny edge detection algorithm is realized by using OpenMP parallel technology. With the addition of parallel task scheduling at the top level, this coarse-grained parallel processing method can realize Gaussian filtering, calculating image gradient, suppression of non-maximum value pixels, and parallel computing of edge points judged by double thresholds. This paper mainly adopts the static scheduling mode, and the specific parallel process: when m CPU cores are allocated to process the image size $H \times H$, each core (or thread) will process $\left(H \times H\right)/m$ image data.

The parallel model of OpenMP is in the form of Fork-Join, and the area between Fork and Join is a parallel region. When the original thread encounters a parallel structure instruction, it creates a thread group and executes the next instruction in parallel, that is, the Fork action. When exiting the parallel structure, only the original thread continues to execute, and the other threads end, that is, the Join action. The OMP_Canny parallel algorithm executes the Fork action to

open the parallel region at the starting position of the Gaussian filtering operation, and executes the Join action to end the parallel region when the detection of all edge points of the image is completed, thus forming the following four parallel regions.

(1) Parallel region of smooth image. First initialize the variable, then convolution the neighborhood of the pixel with the Gaussian filter template, and finally, update the convolution value back to the corresponding position of the image.

(2) The parallel region that determines the amplitude and direction of the image gradient. Firstly, the variables are initialized in x、y direction, and then the neighborhood of the pixel after the Gaussian filter is convoluted with the Sobel filter template in x、y direction, respectively. Finally, the gradient amplitude and gradient direction of the pixel are calculated.

(3) Determine the parallel region of the non-maximum value suppressed gradient image. According to the gradient direction of the pixel, the pixel is suppressed by non-maximum value, and the non-maximum value suppression image is obtained.

(4) Determine the parallel region of image edge points. Using a double threshold algorithm to detect edge points and connect the edges of non-maximum value suppressed images.

5.2 The CUDA_Canny parallel algorithm

According to the parallelism analysis of the Canny algorithm, there are obvious data computational parallelism in image Gaussian filtering, calculating the amplitude and direction of image gradient, non-maximum value suppression gradient image generation, and image edge point detection. The mapping between the image and the execution thread mainly lies in the correspondence between the pixel and the CUDA thread. If the image size is $H \times H$ and the GPU has a streaming multiprocessors, the image data of $H \times H$ size is inputted into the GPU memory. In the software architecture, each GPU streaming multiprocessor contains b thread blocks and each thread block contains c threads, so it can be calculated that each thread can complete the Gaussian filtering processing of $(H \times H)/(a \times b \times c)$ pixels. The parallel processing of the gradient amplitude and gradient direction of each pixel is the same. In the experiment of CUDA_Canny parallel algorithm implementation, the GPU used is GTX 1050 with 24 streaming multiprocessors, each streaming multiprocessor contains 32 thread blocks, and each thread block has 1024 threads.

6 Data testing and result discussion

6.1 Experimental conditions

(1) The hardware platform is built. This experimental scheme uses two different environments with heterogeneous computing capabilities, and the specific hardware configuration information is shown in Table 3.

Table 3 Performance parameters of GPU Computing platform

Configuration number	CPU type	CPU frequency	Memory/GB	GPU type	Video memory	Number of CUDA cores	Number of SM	Number of blocks per SM	Number of threads per block
Configuration 1	Intel Core i7-8700K (six cores)	3.7 GHz	4	Geforce GTX 1050	3 GB GDDR5	768	24	32	1024
Configuration 2	AMD Ryzen 5 3600XT (six cores)	3.8 GHz	4	Radeon RX 560	4 GB GDDR5	896	28	32	1024

(2) The software platform is built. The operating system is Microsoft Window 10 64-bit, the GPU application programming interface is CUDA 10.2, the OpenCL version is AMD

APP SDK 3.0, and the development environment is Microsoft Visual Studio 2017.

6.2 Image quality evaluation

6.2.1 Visual effect comparison

In order to verify the effectiveness of this method, five images are selected as test objects. The resolutions of the images "Star", "Cameraman", "Head CT", "Painting" and "Light-house" are 256 × 256, 256 × 256, 512 × 512, 380 × 375, and 512 × 512, respectively. Four serial/parallel Canny edge detection algorithms are tested and the experimental results are shown in Figure 10.

It can be seen from Figure 10 that the Star result image is completely connected, without disconnection, with good coherence and high definition. The cameraman outline of the Cameraman result image and the lines of the camera bracket are very smooth. In the Head CT result image, the outer contours of the brain (the gray area in the image), the outer contours of the spinal cord, and the outer contours of the head are very clear and very smooth, and there are almost no breakpoints. In the four figures of the Painting result image, the outline of each figure is closed and can be clearly observed. The texture of the exterior wall, the fence, and the edges of the eaves are very clear in the Light-house result image.

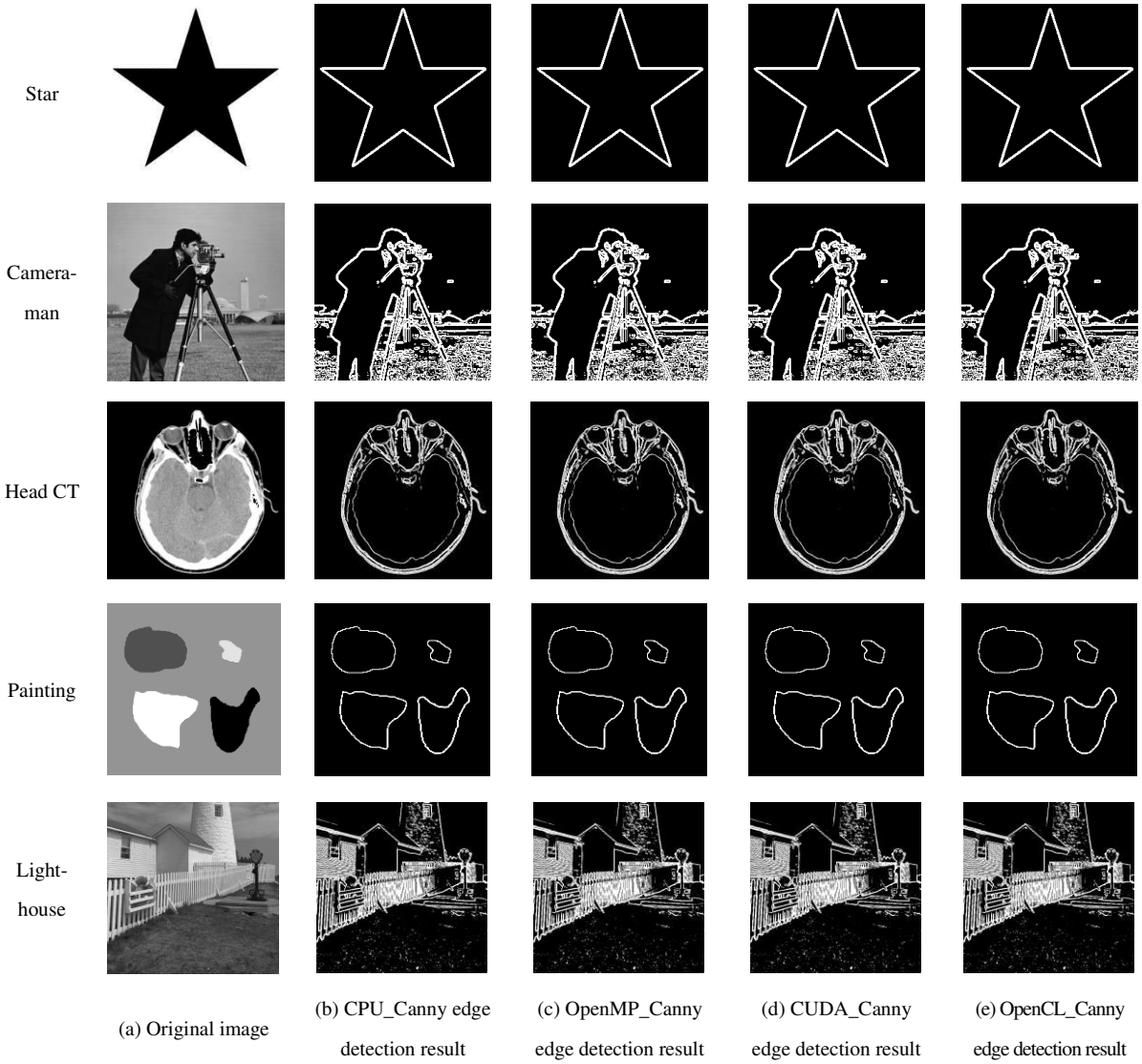

Fig. 10 Edge detection effect images of four different Canny algorithms

It can be seen from Figure 10 that the effects of the serial Canny algorithm and optimized accelerated algorithm are basically the same, and the four edge detection operators can

obtain image edges more accurately. The above experiments show that the OCL_Canny parallel algorithm is feasible.

6.2.2 Comparison of evaluation parameters

In order to evaluate the effect of image edge detection, the average gradient value of the image is selected as the evaluation parameter. The Average Gradient (AG) is also called image sharpness, which is an indicator of the rate of gray change in image. The average gradient is defined as:

$$D_x(i,\ j) = I(i,\ j) - I(i+1,\ j) \tag{6}$$

$$D_y(i,\ j) = I(i,\ j) - I(i,\ j+1) \tag{7}$$

$$AG = \frac{1}{M \times N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \sqrt{\frac{1}{2}\left(D_x^2(i,\ j) + D_y^2(i,\ j)\right)} \tag{8}$$

Among them, $D_x(i,j), D_y(i,j)$ denotes the gradient of the image in the x direction and y direction respectively. $I(i,j)$ is the gray value of the image $(i,\ j)$, $(i,\ j)$ is the position index of the pixel in the image, and the image frame size of the image I is $M \times N$. The image average gradients of different Canny edge detection algorithms are shown in Table 4.

Table 4 Average gradient of Canny edge detection algorithm in different images

Processing method	Star	Camera-man	Head CT	Painting	Lighthouse
No processing	2.79	7.16	5.54	0.88	13.08
CPU_Canny	5.72	34.59	12.09	4.20	30.64
OMP_Canny	5.73	34.59	12.09	4.20	30.64
CUDA_Canny	5.73	34.59	12.11	4.20	30.95
OCL_Canny	5.73	34.59	12.12	4.20	30.95

It can be seen from Table 4 that the average gradient obtained by the OCL_Canny parallel algorithm on the test image set is the largest, indicating that the algorithm in this paper is the best in preserving edge details. At the same time, the average gradient data of the test image under serial/parallel Canny edge extraction are almost the same. It shows that the OCL_Canny parallel algorithm is correct and feasible.

6.3 Analysis of experimental data

6.3.1 Operation time comparison

In order to verify the high performance of the proposed algorithm, nine groups of images of different sizes are selected for experimental analysis. CPU_Canny algorithm, OMP_Canny algorithm, and CUDA_Canny algorithm measured the execution time in the configuration 1 environment, while the OCL_Canny algorithm measured execution time in the configuration 1 and configuration 2 environment, respectively. After many times of execution, the average value of the system is taken as the execution time. The time-consuming statistics are shown in Table 5.

Table 5 Time-consuming comparison of Canny algorithms under different architectures

Image resolution (px)	CPU_Canny (ms)	Parallel time (ms)			
		OMP_Canny	CUDA_Canny	OCL_Canny (AMD)	OCL_Canny (NVIDIA)
256×256	9.45	2.90	3.26	3.14	3.02
512×512	40.12	11.33	4.59	4.41	4.28
1280×720	103.26	27.46	11.02	10.24	9.87
1024×1024	147.69	35.58	12.49	10.68	10.31
1600×1200	309.26	67.67	21.11	19.42	18.34
2048×1536	548.43	112.85	31.83	29.05	28.74
3500×3500	2311.45	459.54	120.29	117.62	115.97
4828×4828	4024.03	762.13	207.96	204.61	199.80
7452×8024	10105.12	1867.84	516.74	513.15	488.64

In order to more intuitively analyze the time characteristics of the Canny algorithm, it is shown in Figure 11. As can be seen from Figure 11, with the continuous increase of the size of nine groups of images, the time-consumption of the Canny algorithm under different computing architectures increases linearly. The time-consuming of the CPU_Canny serial algorithm is gradually approaching to $O\left(H^2n^2\right)$. The experimental results are consistent with the theoretical analysis of time complexity. The time-consuming curve of the Canny algorithm under OpenMP architecture shows a steady upward trend of a small slope. On the other hand, the time-consuming curve of the Canny algorithm under CUDA and OpenCL architecture almost coincides with the horizontal axis in the graph, that is, the time-consuming change of the algorithm is very small with the increase of the amount of data processed.

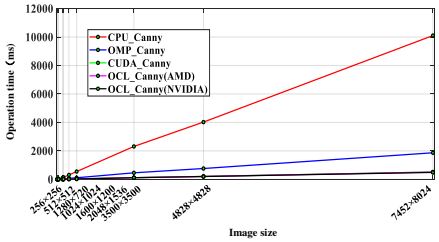

Fig. 11 Time-consuming analysis of the Canny algorithm

Literature [15] and Literature [16] reported the implementation results of the Canny algorithm under CUDA and OpenCL architecture respectively, and literature [11] reported the implementation results of the Canny algorithm under FPGA computing architecture. The data shown in these literatures are compared with the time-consuming of the algorithms in this paper, as shown in Table 6. According to Table 6, the time-consuming Canny serial algorithm implemented in literature [15] and literature [11] on three groups of images is slightly higher than that of the CPU_Canny algorithm in this paper. The time-consuming of the CUDA_Canny and OCL_Canny (NVIDIA) parallel algorithm on three sets of images is significantly lower than that of the CUDA version of the Canny algorithm in the Literature [15]. The time-consuming of the OCL_Canny (NVIDIA) parallel algorithm on four sets of images is significantly lower than that of the OpenCL version of the Canny algorithm in Literature [16]. Therefore, OCL_Canny (NVIDIA) parallel algorithm has the advantage of time-consuming compared with other schemes.

Table 6 Comparison of operation time in related literature

Image resolution (px)	CPU algorithm (ms)			CUDA algorithm (ms)		Literature [11]	OpenCL algorithm (ms)	
	Literature [15]	Literature [11]	CPU_ Canny	Literature [15]	CUDA_ Canny		Literature [16]	OCL_Canny (NVIDIA)
256×256	10.00	—	9.45	5.00	3.26	—	—	3.02
512×512	41.00	78.24	40.12	22.00	4.59	4.61	—	4.28
1280×720	—	—	103.26	—	11.02	—	19.04	9.87
1024×1024	149.00	—	147.69	82.00	12.49	—	—	10.31
1600×1200	—	—	309.26	—	21.11	—	39.46	18.34
2048×1536	—	—	548.43	—	31.83	—	58.03	28.74
3500×3500	—	—	2311.45	—	120.29	—	239.89	115.97

6.3.2 Accelerated performance analysis

(1) Speedup discussion

In order to select a high-performance Canny parallel algorithm, the speedup is used as the performance measure.

Definition 1: speedup S_{OMP} is defined as the time-consuming comparison between the CPU_Canny serial algorithm and the OMP_Canny parallel algorithm. The calculation equation of S_{OMP} is

$$S_{OMP} = \frac{T_{CPU\_Canny}}{T_{OMP\_Canny}} \tag{9}$$

Definition 2: speedup S_{CUDA} is defined as the time-

consuming comparison between the CPU_Canny serial algorithm and the CUDA_Canny parallel algorithm. The calculation equation of S_{CUDA} is

$$S_{CUDA} = \frac{T_{CPU\_Canny}}{T_{CUDA\_Canny}} \tag{10}$$

Definition 3: speedup S_{OCL} is defined as the time-consuming comparison between the CPU_Canny serial algorithm and the OCL_Canny parallel algorithm on the corresponding GPU platform. The calculation equation of S_{OCL} is

$$S_{OCL} = \frac{T_{CPU\_Canny}}{T_{OCL\_Canny}} \tag{11}$$

Definition 4: relative speedup $RS_{OMP-OCL}$ is defined as the time-consuming comparison between the OMP_Canny parallel algorithm and the NVIDIA GPU-based OCL_Canny parallel algorithm. The calculation equation of $RS_{OMP-OCL}$ is

$$RS_{OMP-OCL} = \frac{T_{OMP\_Canny}}{T_{OCL\_Canny}} \tag{12}$$

Definition 5: relative speedup $RS_{CUDA-OCL}$ is defined as the time-consuming comparison between the CUDA _ Canny parallel algorithm and the NVIDIA GPU-based OCL_Canny parallel algorithm. The calculation equation of $RS_{CUDA-OCL}$ is

$$RS_{CUDA-OCL} = \frac{T_{CUDA\_Canny}}{T_{OCL\_Canny}} \tag{13}$$

The speedup achieved by the OMP_Canny, CUDA_Canny, and OCL_Canny parallel algorithms on each group of test images is shown in Table 7.

Table 7 Acceleration effect of the Canny algorithm on different platforms

Image resolution	Speedup				Relative speedup	
(px)	S_{OMP}	S_{CUDA}	S_{OCL} (AMD)	S_{OCL} (NVIDIA)	$RS_{OMP-OCL}$	$RS_{CUDA-OCL}$
256×256	3.26	2.90	3.01	3.13	0.96	1.08
512×512	3.54	8.74	9.10	9.37	2.65	1.07
1280×720	3.76	9.37	10.08	10.46	2.78	1.12
1024×1024	4.15	11.82	13.83	14.32	3.45	1.21
1600×1200	4.57	14.65	15.92	16.86	3.69	1.15
2048×1536	4.86	17.23	18.88	19.08	3.93	1.11
3500×3500	5.03	19.22	19.65	19.93	3.96	1.04
4828×4828	5.28	19.35	19.67	20.14	3.81	1.04
7452×8024	5.41	19.56	19.69	20.68	3.82	1.06

Figure 12 shows the speedup change of the Canny parallel algorithm under different image data sizes. Under different parallel computing architectures, the Canny algorithm achieves a certain speedup. With the increase of image resolution, S_{OMP} gradually becomes larger, indicating that the acceleration effect of the OMP_Canny parallel algorithm is more obvious when dealing with large images. When the image resolution is low, the acceleration effect of the CUDA_Canny and OCL_Canny parallel algorithms is not obvious. Because GPU computing needs to transfer computing data through a low-speed PCI-E bus, and the number of work-items started is not enough to hide the time overhead of data transfer and kernel function startup, that is, the performance improvement brought by many-core computing cannot offset the additional communication and function startup time overhead brought by heterogeneous architecture. With the increase of image resolution, the computation shifts from I/O-intensive to computing-intensive. When the image resolution is less than 2048 × 1536, the speedup of the OCL_Canny parallel algorithm increases faster. However, when the image resolution exceeds 2048 × 1536, the slope of the S_{OCL} (NVIDIA) curve gradually smooths and tends to be stable, and the OCL_Canny parallel algorithm achieves a speedup of 20.68 times.

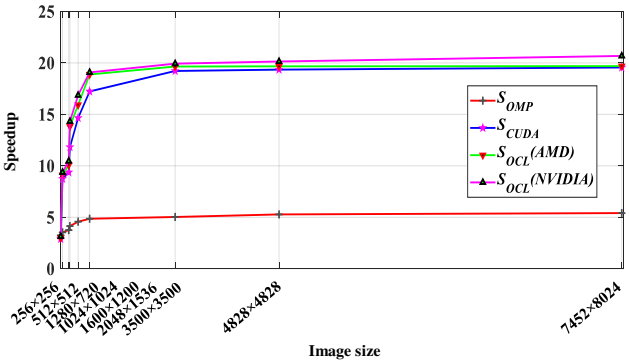

Fig. 12 Performance acceleration of the Canny algorithm

Table 8 shows the acceleration effect of CUDA_Canny and OCL_Canny parallel algorithms and related literature on three groups of images. As can be seen from the table, when dealing with small images, the acceleration effect of the data in Literature [15] is similar to that of the OCL_Canny parallel algorithm. With the expansion of the image frame, the growth rate of S_{CUDA} and S_{OCL} (NVIDIA) is faster than that of Literature [15], indicating that the OCL_Canny parallel algorithm is more suitable for the fast processing of large images than Literature [15].

Table 8 Comparison of acceleration ratio of related literature

Image resolution	Speedup		
(px)	Literature [15]	S_{CUDA}	S_{OCL} (NVIDIA)
256×256	2.00	2.90	3.13
512×512	1.86	8.74	9.37
1024×1024	1.81	11.82	14.32

Figure 13 visually shows the performance comparison among the three parallel algorithms OMP_Canny, CUDA_Canny, and OCL_Canny. As can be seen from Figure 13, when the image is small, the OCL_Canny parallel algorithm has no obvious performance advantage over the OMP_Canny parallel algorithm. The OCL_Canny parallel algorithm needs data exchange between memory and video memory, which degrades the performance of the OCL_Canny parallel algorithm. However, when the image is larger, the number of work-items started is more, the proportion of kernel function execution time is reduced, and the large value $RS_{OMP\text{-}OCL}$ reflects the strong data processing ability of the GPU. The acceleration ability of the CUDA_Canny and OCL_Canny parallel algorithms is basically the same and $RS_{CUDA\text{-}OCL}$ achieves a maximum acceleration advantage of 1.21 times.

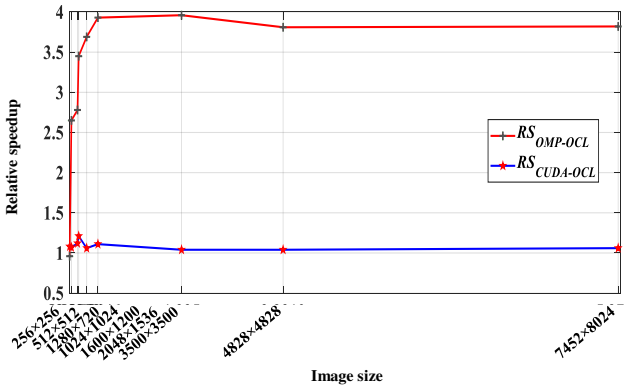

Fig. 13 Performance comparison between different parallel Canny algorithms

(2) Discussion on portability of the OCL_Canny parallel algorithm

As can be seen from Figure 8, the OCL_Canny parallel algorithm has a good acceleration effect on different GPU platforms. At the same time, the values of S_{OCL}(AMD) and S_{OCL}(NVIDIA) are very similar in nine groups of images with different image sizes. It shows that the OCL_Canny parallel algorithm has good platform scalability and data scalability.

6.3.3 System bottleneck analysis

In the operation and execution of the OCL_Canny algorithm based on GPU acceleration, there are a large number of memory read and write operations in the processing steps of Gaussian filtering, image gradient calculation, image non-maximum value suppression, and edge detection. According to the previous analysis, in the kernel operation of the Gaussian filter, the system needs to read $H^2 \times n^2$ times and write H^2 times to the extended image. In calling the kernel operation to calculate the image gradient, it is necessary to read data $H^2 \times n^2$ times for the extended image and write data $2 \times H^2$ times for the amplitude and direction of the image gradient. In calling the kernel operation of the non-maximum value suppression of the image, it is necessary to read data $2 \times H^2$ times for the amplitude and direction of the image gradient and write data H^2 times for the original image. In calling the kernel operation of edge detection, it is necessary

to read data H^2 times for the amplitude of the image gradient and write data H^2 times for the original image. Therefore, in the operation and execution of the OCL_Canny algorithm, a total of $2\times H^2\times n^2+8\times H^2$ memory data are needed to read and write. Suppose, the image resolution is 2048 × 1536, the size of the filter template is 3 × 3, and each pixel takes up 4 B storage space. According to the calculation, the total amount of image data accessed by the OCL_Canny system is about 0.3 GB. The total amount of image data divided by the running time of the kernel 4.81 ms, which shows that the bandwidth of the OCL_Canny system is about 62.37 GB/s. At this point, the actual bandwidth of the system is close to the bandwidth 84 GB/s of GeForce GTX 1050. Therefore, the global memory bandwidth has become the main performance bottleneck of the OCL_Canny system.

7 Conclusion

With the rapid development of GPU, GPU is used more and more widely, and the advantage of GPU parallel computing is increasing day by day. At the same time, the requirements for the performance and optimization of parallel computing are getting higher and higher. Through the research on the parallel transplantation and optimization of the Canny edge detection algorithm, this paper puts forward the following three suggestions: (1) For large-scale computing-intensive tasks, the performance of the algorithm can be improved through the parallel computing of the GPU. At the same time, the overall performance can be improved through the cooperation of heterogeneous platforms GPU and CPU. (2) Memory access optimization plays an important role in improving the performance of the overall algorithm. Therefore, the efficiency of memory access can be improved by means of vectorization, data localization, and fine tuning. (3) In order to achieve efficient mapping between threads and the underlying hardware, it is necessary to consider the characteristics of hardware architecture and image processing algorithms, and use several optimization strategies to achieve high-performance algorithms. The experimental results show that the OCL_Canny parallel algorithm achieves a performance speedup of 3.13 times ~ 20.68 times under different image data sizes. It provides a theoretical basis for other image processing algorithms and improves the engineering application value of the image edge detection algorithm. In the next step, the bandwidth bottleneck problem in the image processing algorithm will be studied to further improve the performance of the algorithm.

Author contributions YS: Writing-review & editing, Software, Investigation, Visualization. CL: Conceptualization, Writing-review & editing, Funding acquisition, Supervision. QZ: Methodology, Formal analysis, Software, Investigation, Writing-review & editing. HX: Writing-original draft, Writing & editing, Software, Investigation.

Funding This work was supported by the National Natural Science Foundation of China (Nos. 61572444, 61250007), the Key Scientific Research Projects of Henan Province Colleges and Universities of China (No. 22A520049), and the Natural Science Foundation of Shandong Province (No. ZR2022MD039).

Data availability The datasets generated or analyzed during this study are available from the corresponding author on reasonable request.

Declarations

Conflict of interest The authors declare that they have no conflicts of interest to report regarding the present study.

References

1. Taslimi, S., Faraji, R., Aghasi, A., et al.: Adaptive edge detection technique implemented on FPGA. Iranian Journal of Science and Technology-Transactions of Electrical Engineering. **44**(4), 1571−1582 (2020)
2. Morar, A., Moldoveanu, F., Asavei, V., et al.: Multi-GPGPU based medical image processing in hip replacement. Control Eng Appl Inf. **14**(3), 25–34 (2012)
3. Dhivya, R., Prakash, R.,: Edge detection of satellite image using fuzzy logic. Cluster Comput. **22**(5), 11891–11898 (2019)
4. Al Badawi, A., Veeravalli, B., Lin, J., et al.: Multi-GPU design and performance evaluation of homomorphic encryption on GPU clusters. IEEE T Parall Distr. **32**(2), 379–391 (2021)
5. Wisultschew, C., Perez, A., Otero, A., et al.: Characterizing deep neural networks on edge computing systems for object classification in 3D point clouds. IEEE Sens J. **22**(17), 17075–17089 (2022)
6. Liu, X.X., Mao, M.J., Bi, X.Y., et al.: Exploring applications of STT-RAM in GPU architectures. IEEE T Circuits-I. **68**(1), 238–249 (2021)

7. Canny, J.: A computational approach to edge detection. IEEE T Pattern Anal. **8**(6), 679–698 (1986)
8. Wachowicz, A., Pytlik, J., Malysiak-Mrozek, B., et al.: Edge computing in IoT-enabled honeybee monitoring for the detection of varroa destructor. INT J Ap Mat Com-Pol. **32**(3), 355–369 (2022)
9. Risso, M., Burrello, A., Conti, F., et al.: Lightweight neural architecture search for temporal convolutional networks at the edge. IEEE T Comput. **72**(3), 744–758 (2023)
10. Weizhong, S., Weiwei, C., Yanming, F., et al.: FPGA-based real-time edge detection and its implementation for deep-space images. electronic science and technology. **33**(5), 45−49 (2020)
11. Jin, W., Jun, Z., Cong, L., et al.: Implementation of SDSo C acceleration algorithm for edge detection algorithm in machine vision. Computer Engineering and Applications. **55**(12), 208−214 (2019)
12. Keqiang, X., Guangming, L., Renren, L., et al.: Implemention and optimization of Canny operator on DSP. Modern Electronics Technique. **37**(6), 8−11 (2014)
13. Xiangjiao，L.,Guangliang，L.,Xuewu，Z.,et al.: The parallel canny algorithm based on TBB. Journal of Nanyang Institute of Technology. **6**(3), 47−50 (2014)
14. Yue, Z., Xiaohong, W., Xiaohai, He.: Real-time image edge detection based on GPU. Electronic Measurement Technology. **31**(2), 140−142 (2009)
15. Bin, T., Wen, L.: Fast Canny algorithm based on GPU+CPU. Chinese Journal of Liquid Crystals and Displays. **31**(7), 714–720 (2016)
16. Jin, W., Ying, L., Zhen-tao, L., et al.: GPU implementation of machine vision algorithm based on OpenCL. Computer Engineering and Design. **40**(2), 346−351 (2019)
17. Iqbal, B., Iqbal, W., Khan, N., et al.: Canny edge detection and Hough transform for high resolution video streams using Hadoop and Spark. Cluster Comput. **23**(1), 397–408 (2020)
18. Cao, J.F., Chen, L.C., Wang, M., et al.: Implementing a parallel image edge detection algorithm based on the Otsu-Canny operator on the Hadoop platform. Comput Intel Neurosc. (03), 1–13 (2018)
19. Xiaoli, H., Li, D., Jie, J.: Real-time image edge detection of the improved Canny algorithm. Journal of Inner Mongolia University of Science and Technology. **34**(3), 262−266 (2015)
20. Sangeetha, D., Deepa, P.: FPGA implementation of cost-effective robust Canny edge detection algorithm. J Real-Time Image Pr. **16**(4), 957–970 (2019)
21. Lee, J., Tang, H., Park, J.: Energy efficient Canny edge detector for advanced mobile vision applications. IEEE T Circ Syst Vid. **28**(4), 1037–1046 (2018)
22. Suwen, Z., Zhixing, C., Yixin, S.U.: Improved Canny edge detection algorithm and implementation in FPGA. Infrared Technology. **32**(2), 93–96 (2010)
23. Shengxiao, N., Sheng, W., Jingjing, Y.: A Fast image segmentation algorithm fully based on edge information. Journal of Computer-Aided Design and Computer Graphics. **24**(11), 1410−1419 (2012)
24. Fuqiang, Z., Cao, Y., Wang, X.M.: Fast and resource-efficient hardware implementation of modified line segment detector. IEEE T Circ Syst Vid. **28**(11), 3262–3273 (2018)
25. Sivakumar, V., Janakiraman, N.: A novel method for segmenting brain tumor using modified watershed algorithm in MRI image with FPGA. Biosystems. **198**(S1), 1–13 (2020)
26. Hongye, Z.: Optimization identification and simulation about household registration management personal fingerprint image. Heilongjiang Science. **11**(12), 1−3 (2020)
27. Rongbao, C., Tianze, F., Honghu, Jiang.: Identification method of welding perpendicularity for components based on DSP+FPGA. Computer Measurement and Control. **25**(6), 207−210, 214 (2017)
28. Hanjun, Jin., Zeng, T.: Contour extraction of moving objects in video sequences based on GPU. Electronic Measurement Technology. **39**(11), 85−88 (2016)
29. Tengzhang, J., Yuxin, H., Peng, L., et al.: A method of multi-ship target detection and tracking by on-orbit satellite. Journal of University of Chinese Academy of Sciences. **37**(3), 368−378 (2020)
30. Gadowski, S., Tomiczak, K., Komsta, L.: High dynamic range in video densitometry-a comparative study to classic video scanning on Gentiana extracts. JPC-J Planar Chromat. **36**(1), 3–8 (2023)
31. Alvarez-Farre, X., Gorobets, A., Trias, F.X.: A hierarchical parallel implementation for heterogeneous computing. Application to algebra-based CFD simulations on hybrid supercomputers. Comput Fluids. **214**(10), 1–10 (2021)
32. Banas, K., Kruzel, F., Bielanski, J.: Optimal kernel design for finite-element numerical integration on GPUs. Comput Sci Eng. **22**(6), 61–74 (2020)
33. Tran, T.H., Sun, K.C., Simon, S.: A GPU-accelerated light-field super-resolution framework based on mixed noise model and

weighted regularization. J Real-Time Image Pr. **19**(5), 893–910 (2022)

34. Simmross-Wattenberg, F., Rodríguez-Cayetano, M., Royuela-del-Val, J., et al.: OpenCLIPER: An OpenCL-based C++ framework for overhead-reduced medical image processing and reconstruction on heterogeneous devices. IEEE J Biomed Health. **23**(4), 1702–1709 (2019)
35. Xiao, H., Fan, Y.M., Ge, F., et al.: Algorithm-hardware co-design of real-time edge detection for deep-space autonomous optical navigation. IEICE T Inf Syst. **E103D**(10), 2047–2058 (2020)
36. Zimu, X., Ki-Young, S., M.Gupta, Madan.: Development of a CNN edge detection model of noised X-ray images for enhanced performance of non-destructive testing. Measurement. **174**(10), 1–17 (2021)
37. Lee, D.H.E., Chen, P.Y., Yang, F.H., et al.: High-efficient low-cost VLSI implementation for Canny edge detection. J Inf Sci Eng. **36**(3), 535–546 (2020)
38. Lakshmi, S.J., Deepa, P.: Blind image deblurring using GLCM and negans obtuse mono proximate distance. Imaging Sci J. **70**(01), 19–29 (2023)
39. Chen, J.Y., Xi, Z.H., Wei, C., et al.: Multiple object tracking using edge multi-channel gradient model with ORB feature. IEEE Access. **9**(2), 2294–2309 (2021)
40. Zhang, X., Lu, W., Ding, Y.W., et al.: A mixed method for feature extraction based on resonance filtering. intelligent automation and soft computing. **35**(03), 3141–3154 (2022)

Yupu Song received her M.D. degree in software engineering from Beijing University of Technology, China in 2002. She is currently an associate professor in the Department of Computer, Shangqiu Polytechnic. Her main research interests include data analysis and parallel computing.

Cailin Li received the Ph.D. degree in photogrammetry and remote sensing from the School of Remote Sensing and Information Engineering, Wuhan University, China, in 2011. He is currently an associate professor in the School of Civil and Architectural Engineering, Shandong University of Technology, China. His main research interests include digital photogrammetry and computer vision and digital image processing.

Qinglei Zhou received the Ph.D. degree in software and theory from Xi'an Jiaotong University, China, in 2002. He is currently a professor and doctoral supervisor in the School of Computer and Artificial Intelligence, Zhengzhou University, China. His main research interests include parallel algorithm, image processing, and parallel computing.

Han Xiao received the Ph.D. degree in photogrammetry and remote sensing from the School of Remote Sensing and Information Engineering, Wuhan University, China, in 2011. From 2011 to 2014, he was a Postdoctoral Researcher with School of Information Engineering, Zhengzhou University, China. Since 2012, he has been a professor level 3 with the School of Information Science and Technology, Zhengzhou Normal University, China. His main research interests include research and design of massively parallel algorithms, research on parallel processing of remote sensing big data, photogrammetry and remote sensing, and parallel computing.