Received March 1, 2021, accepted March 12, 2021, date of publication March 24, 2021, date of current version May 5, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3068772

A Parallel Algorithm of Image Mean Filtering Based on OpenCL

HAN XIAO[1], BAOYUN GUO[2], HONGYAN ZHANG[1], AND CAILIN LI[2]
[1]School of Information Science and Technology, Zhengzhou Normal University, Zhengzhou 450044, China
[2]School of Civil and Architectural Engineering, Shandong University of Technology, Zibo 255000, China

Corresponding author: Cailin Li (licailin@sdut.edu.cn)

This work was supported in part by the Key Laboratory Open Foundation for Geo-Environmental Monitoring of Great Bay Area (Shenzhen University) through the Ministry of Natural Resources of the People's Republic of China under Grant SZU51029202003, in part by the National Natural Science Foundation of China under Grant 41701525 and Grant 41601496, and in part by the Key project of Art Science in Shandong Province under Grant ZD202008267 and Grant 201806353.

ABSTRACT The image will be contaminated by noise during the imaging process, which severely degrades the image quality. It is necessary to filter the collected image. With the increasing amount of image data, the traditional single-processor or multiprocessor computing equipment has been unable to meet the requirements of real-time data processing. In this paper, the computational model of weighted mean filtering and the characteristics of high performance computer architecture are studied. An efficient hierarchical image weighted mean filtering parallel algorithm for Open Computing Language (OpenCL) is designed and implemented, which can fully express the parallelism of the computing model. The parallel algorithm takes full account of the characteristics of image discrete convolution computing and the multi-layer logic architecture of high performance computer, deeply excavates the parallelism of the computing platform and computing model, and realizes the efficient task mapping from computing model to computing resources. The model is implemented in parallel with the two levels of work-group and work-item. The experimental results show that compared with the serial algorithm based on CPU, the parallel algorithm based on Open Multi-Processing (OpenMP) and the parallel algorithm based on Compute Unified Device Architecture (CUDA), the parallel algorithm of weighted mean filtering achieves 20.88 times, 18.52 times and 1.26 times acceleration ratio on the NVIDIA GPU computing platform based on OpenCL architecture, respectively. It realizes better computing performance and runs on different Graphic Processing Unit (GPU) computing platforms, and has good portability and scalability.

INDEX TERMS Weighted mean filtering, Gaussian noise, Graphic Processing Unit (GPU), Open Computing Language (OpenCL), parallel algorithm.

I. INTRODUCTION

In the process of image acquisition, transmission, storage, and processing, due to the performance limitations of transmission media and receiving equipment, there are inevitably external interference and internal interference [1]. This leads to the generation of all kinds of noise, resulting in varying degrees of degradation in the image quality, and the image becomes blurred. The effect of removing image noise is an important part of image analysis and image recognition [2]. Therefore, in order to effectively deal with noise and improve image quality, image filtering technology has become a research focus in the fields of image segmentation, feature extraction, pattern recognition, and so on. Gaussian noise is the most common noise in digital images [3]. Mean filtering has the characteristics of simple operation and strong ability to remove Gaussian noise, which makes it the most commonly used method among many denoising methods. However, mean filtering also destroys the details of the image while the image is denoising so that the image becomes blurred. Mean filtering mainly includes arithmetic (weighted) mean filtering, geometric mean filtering, harmonic mean filtering, and inverse harmonic mean filtering, and so on [4]. Because the amount of image data that needs to be processed is getting larger and larger, and the complicated filtering processing is required to obtain clear images. At the same time, some application fields have higher real-time requirements for the process of obtaining clear images, which

The associate editor coordinating the review of this manuscript and approving it for publication was Qiangqiang Yuan.

makes high-speed image processing technology become a key technology [5].

At present, CPU-based personal computers, workstations, and large computing servers are usually used for image processing. At this time, a large amount of money needs to be invested to purchase a number of computing equipment with sufficient processing capacity. The software is implemented by parallel programming techniques such as Open Multi-Processing (OpenMP) or Message Passing Interface (MPI). If it is implemented by Field Programmable Gate Array (FPGA) or Digital Signal Processor (DSP), more complex programming methods and expensive hardware equipment are required [6]. Modern Graphic Processing Unit (GPU) uses relatively simple control logic, and a large number of transistors are used in Arithmetic Logic Unit (ALU) to participate in the data processing. With its strong computing power and excellent performance-to-price ratio, GPU has attracted more and more general-purpose computing, including fluid simulation, video detection, sequence alignment, and protein molecular field, and so on [7]. In order to overcome the shortcomings of traditional GPU programming based on the graphical interface, AMD and NVIDIA respectively put forward their programming models-Brook+ and Compute Unified Device Architecture (CUDA). However, AMD GPU is not compatible with NVIDIA GPU, and the programming model of GPU is mostly limited to a fixed platform, which brings a lot of inconvenience to the design and transplantation of the program. In order to give full play to the performance potential of various devices under heterogeneous processing platforms, and make the software system portable, Khronos Group introduces Open Computing Language (OpenCL). The release of the OpenCL 1.0 standard has brought about new changes. It is an open standard that has been supported by many vendors and can be widely used for general-purpose parallel programming on multi-core CPU, GPU, and other processors [8], [9].

On the basis of comparing and analyzing the similarities and differences of the current main hardware architecture, this paper proposes an OpenCL-based image mean filtering (OCL_MF) parallel algorithm for CPU + GPU heterogeneous computing system. The parallel algorithm of image mean filtering based on OpenMP (OMP_MF) and the parallel algorithm of image mean filtering based on CUDA (CUDA_MF) are compared and studied. Evaluate the acceleration performance of each algorithm and the performance bottleneck of the OCL_MF parallel algorithm, optimize the memory and the NDRange index space configuration, and effectively improve the performance of the algorithm. The optimized algorithm can not only greatly improve the performance, but also achieve cross-platform applications. Specifically, this paper makes the following contributions:

(1) The OCL_MF parallel algorithm for OpenCL heterogeneous systems is proposed, which can efficiently filter images of more than tens of millions of scales on a single heterogeneous node. For the four sets of images in the experiment, the OCL_MF algorithm on a single computer is 20.88 times faster than the CPU serial processing mean filtering algorithm (CPU_MF), with obvious performance advantages.

(2) Adopt a diversified performance comparison standard. This paper implements the processing of image mean filtering on multiple parallel computing platforms, and tests the impact of three parallel modes on the performance of the algorithm. The performance of the OCL_MF parallel algorithm is compared with the OMP_MF algorithm, the CUDA_MF algorithm, and the related literature algorithms. From the performance comparison of the horizontal and vertical directions, we can see that the OCL_MF parallel algorithm has achieved a better performance improvement.

The second section of this paper introduces the related work, the third section studies and analyzes the weighted mean filtering algorithm, the fourth section introduces the design and implementation of OpenCL accelerated weighted mean filtering parallel algorithm, and the fifth section introduces the experimental results and analysis. Finally, a summary of the full text is made.

II. RELATED RESEARCH WORKS

Around the performance improvement method of image denoising, many scholars have made a lot of algorithm improvements, especially in parallel computing. Jaime *et al.* [10] determined the covariance matrix from the partial covariance matrix in parallel, which was used for the fast calculation of the covariance matrix and was convenient for hyperspectral imaging. Zeinab and Gholamreza [11] studied the unsupervised segmentation algorithm of SAR images based on Gabor filter bank and unsupervised spectral regression, and the running time was significantly shortened. Dariusz [12] implemented a recurrent Gaussian filter for separating the shape, waviness, and roughness components of surface texture by using the parallel method. Jorge *et al.* [13] proposed an ultra-low-power massively parallel processing array for image enhancement and edge detection. By optimizing the calculation process of mean filtering, Bai *et al.* [14] realized the de-fogging algorithm on multi-core DSP, which greatly shortened the execution time of the algorithm. Dang and Tsutomu [15] proposed an FPGA image segmentation system based on the Mean Shift algorithm, which reduces the running time of the system. The adaptive transverse LMS, GAL, and QRD-LSL filtering algorithms proposed by Lee *et al.* [16] achieved maximum speed increases of 174%, 432%, and 35.5%, respectively, on computers with four-core and four-way SIMD architecture. He *et al.* [17] proposed an improved mean filtering algorithm to remove image noise quickly and efficiently, which reduced the time cost of the whole algorithm. Devrim [18] implements an adaptive image noise filter based on TDLMS on a multicore computer using OpenMP. Cheng *et al.* [19] parallelized the mean filtering algorithm using multi-core processors and applied it to the MCMC algorithm to improve the running speed of the system. Salvatore *et al.* [20] proposed a full 3D non-local mean parallel method based on multi-GPUs, which had high applicability and scalability.

Nguyen *et al.* [21] implemented a non-local mean denoising filter based on multi-GPUs, which reduced shared memory access and improved denoising speed. Zou *et al.* [22] used OpenMP on a multi-core computer to realize the parallel simulation of the image processing process of ocean wave sample data set by weighted mean filtering, and the acceleration ratio was up to 24.29 times. Subhra *et al.* [23] proposed GPU accelerated particle filter algorithm and obtained 4 times the speedup. Chang *et al.* [24] studied the GPU accelerated bilateral filter based on the BPN model related to image texture features, and obtained 208 acceleration ratio. Chen *et al.* [25] realized the process of image mean filtering using GPU, which greatly reduced the execution time of the algorithm in the linear binocular cost aggregation algorithm. He and Zhang [26] realized the automatic gain compensation algorithm through the GPU parallel mean filtering process, and the speed was increased by about 267 times. Lu *et al.* [27] implemented an inverse Gaussian bilateral filter on GPU, which was applied to a non-photorealistic real-time virtual engraving system. Chang *et al.* [28] proposed a bilateral filter based on CUDA and established a ''high precision'' parameter prediction system, which improved the processing speed. Duan and Li [29] proposed an improved GPU parallel algorithm for image mean filtering based on CUDA, and the acceleration ratio was more than 300×. Xia *et al.* [30] realized the application of image mean filtering algorithm in steel plate image denoising by using CUDA, and obtained 3.91 times acceleration ratio.

At present, most of the research work focuses on using the traditional parallel computing method to parallelize the mean filtering algorithm and apply it to various application fields or carry out parallel processing for the improved mean filtering algorithm. Thus, the processing speed of the application system is improved and the effect of image noise reduction is improved. Most of them use FPGA, DSP, multi-core CPU, CPU cluster, and CUDA many-core processor to realize the parallel algorithm of image processing. Generally speaking, although these studies improve the processing speed of the system, there are some problems, such as complex design, long development cycle, low flexibility, and so on. There is a lack of research results in a single parallel algorithm for image weighted mean filtering, and there is a lack of research on the implementation of cross-platform transplantation to greatly improve the computational efficiency and performance of the algorithm. For this reason, a weighted mean filtering parallel algorithm based on OpenCL architecture is proposed in this paper. The parallelization process of the algorithm retains the data flow structure of the original algorithm. This paper mainly focuses on the data parallelization and task parallelization for the image local processing function which consumes a lot of time in the weighted mean filtering algorithm. Then the performance of the algorithm is further optimized from the aspects of data memory access and kernel configuration of the general parallel computing architecture OpenCL. The performance migration of the algorithm on different GPU computing platforms is realized, and the execution time of the algorithm is effectively shortened.

III. ALGORITHM ANALYSIS

A. OpenCL PARALLEL COMPUTING PLATFORM

In June 2008, Apple proposed the open parallel programming specification OpenCL at the WWDC (World Wide Developers Conference) conference, which was maintained by the Khronos Compute Working Group. On December 9, 2008, at the 2008 Asian SIGGRAPH Conference held in Singapore, the Khronos Group, which is composed of the world's major semiconductor giants, issued the OpenCL version 1.0 technical specification. Apple, AMD, NVIDIA, and IBM have successively announced full support for OpenCL. Version 1.1 of OpenCL was released on June 11, 2010. OpenCL 1.1 is backward compatible with OpenCL 1.0, provides more new features, and improves performance. OpenCL 1.1 adds a large number of functions to improve the flexibility, functionality, and execution efficiency of parallel computing. Version 1.2 of OpenCL was released on November 14, 2011. OpenCL 1.2 adds separate compilation and linking of programs. Custom devices and built-in kernels are supported. Device partitioning allows a device to be partitioned based on a number of partitioning schemes supported by the device. Version 2.0 of OpenCL was released on March 18, 2014. OpenCL 2.0 adds shared virtual memory, enqueue kernels on the device, pipes mechanism, and dynamic parallelism of kernel. OpenCL 2.0 adds that images support for 2D image from buffer, depth images, and sRGB images. Version 2.1 of OpenCL was released on November 5, 2015. The SPIR-V and OpenCL SPIR-V Environment specifications have been added. Version 2.2 of OpenCL was released on July 19, 2019. OpenCL 2.2 adds the third prerequisite (executing non-trivial constructors for program scope global variables). Version 3.0 of OpenCL was released on September 30, 2020. OpenCL 3.0 adds a new API to register a function that will be called when a context is destroyed, enabling an application to safely free user data associated with a context callback function. OpenCL 3.0 adds two new APIs to support creating buffer and image memory objects with additional properties. OpenCL 3.0 adds new queries for the properties arrays specified when creating buffers, images, pipes, samplers, and command queues. OpenCL 3.0 adds new queries to determine supported OpenCL C language versions and supported OpenCL C features.

Figure 1 shows the abstract model defined by OpenCL. In the OpenCL execution architecture, the host-side program is used to uniformly manage and schedule multiple computing devices that support OpenCL [31]. When the host side submits the kernel to the computing device, OpenCL defines the organizational structure of the work-item through the index space and defines how the kernel operates on the computing device in a mapping manner on the computing device [32], [33].

In the OpenCL abstract model, each instance of the execution kernel is called a work-item, which is represented by

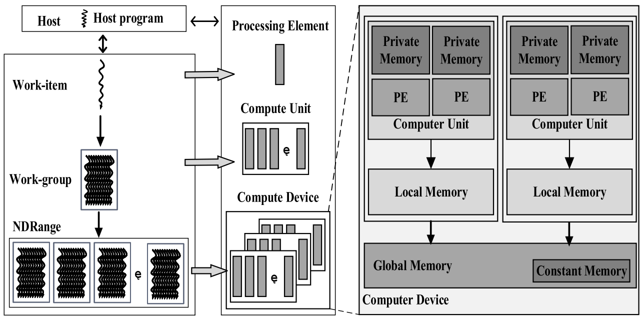


FIGURE 1. OpenCL abstract model.

its coordinates in the NDRange. The corresponding hardware is the processing element [34]. Multiple work-items are organized as a work-group, providing a coarser division of NDRange, where work-items in a given work-group are executed concurrently on the processing element of a compute unit. Global memory and constant memory can be shared between one or more devices within a context, and an OpenCL device is associated with local memory and private memory. Fig. 1 describes the operating space of the various memory areas, that is, their position in the platform [35], [36].

B. ALGORITHM BASIC PROCESS

1) WEIGHTED MEAN FILTER PRINCIPLE

The weighted mean filter is a common technique of linear filter, which has a good effect on suppressing Gaussian noise [37]. The basic idea of the weighted mean filtering algorithm is to replace the gray value of each pixel with the average gray level of several neighborhood pixels. The neighborhood is selected as an 8 neighborhood composed of $\sqrt{2}$ unit distance Δx as the radius r, such as Figure 2 shows the circular region, where $r = \sqrt{\Delta 2x}$ [38], [39].

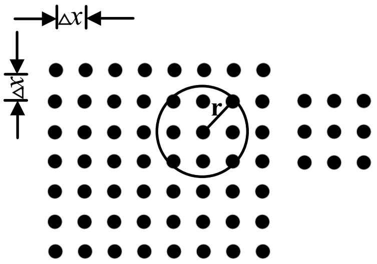


FIGURE 2. Schematic diagram of 8 neighborhood selection of mean filtering.

The weighted mean filtering algorithm is an effective filtering algorithm to remove Gaussian noise. The gray value of the center point of the filtering window is obtained by the weighted average of the gray value of each pixel sample point in the window. The expression is as follows:

$$g(x, y) = \frac{\sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) f(x+s, y+t)}{\sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t)} \tag{1}$$

$$\frac{1}{16} \times \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

FIGURE 3. 3 × 3 weighted mean filter template.

where $f(x+s, y+t)$ is the gray value of the pixel in the neighborhood of the central point (x, y), $g(x, y)$ is the gray estimated value of the filtered central pixel, and $w(s, t)$ is the weight corresponding to the pixel $f(x+s, y+t)$ in the filtering window [40].

When the image polluted by Gaussian noise is filtered by the weighted mean filter, the singular points caused by noise in the smooth region are usually isolated or discontinuous, and the edge of the image is not isolated because of its persistence in a certain direction. Therefore, when we want to estimate the edge points of the image, we want to give the points on the edge of the neighborhood a larger weight, while the non-edge points in the neighborhood correspond to a smaller weight [41]. In the point estimation of the smooth region, because the gray value of the pixel in the neighborhood is similar, the corresponding weight of the pixel which is not polluted by noise or the pollution is not serious is larger, and the weight of the isolated point seriously polluted by noise will be relatively small, so as to smooth the noise. The 3 × 3 smoothing filter shown in Figure 3 is used in this paper [42].

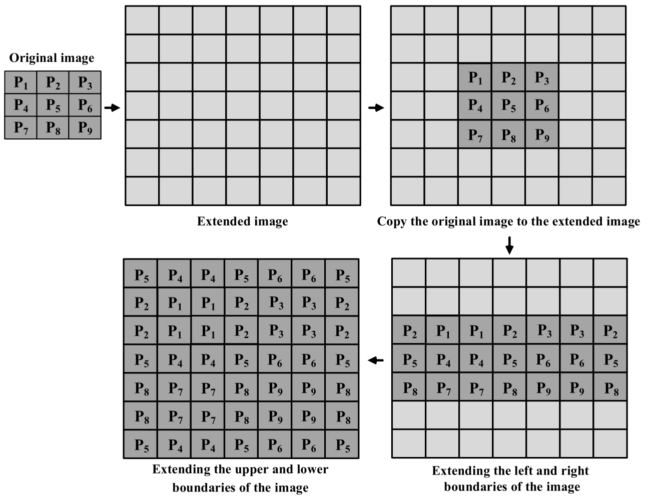

Original image
Extended image
Copy the original image to the extended image
Extending the upper and lower boundaries of the image
Extending the left and right boundaries of the image

FIGURE 4. Image extension principle.

2) IMAGE BOUNDARY PROCESSING

In the image processing algorithm, a large number of algorithms use templates to participate in the calculation. These algorithms have the same characteristics: when dealing with the current pixels, they need the information of other pixels in the neighborhood within the scope of the template. Therefore, when executing these algorithms, it is necessary to traverse the image pixels corresponding to the template according to the size of the template. When the template is used to traverse the image, the judgment is more complex in the boundary area of the image. Because at this point, the area covered by the template may have exceeded the boundaries of the image [43].

At present, the method of filling 0 or retaining the original value is mostly used for the extended boundary. The weighted mean filter parallel algorithm uses even extended explicit edge expansion method for image expansion, that is, the 0-th column is copied to the first column of the left extension, and the first column is copied to the second column of the left extension, and so on, the extension of the four boundaries [44]. In this method, the original image is extended by using the points close to the edge pixels, and the similarity is higher. Taking the 3×3 image size and the template of the 5×5 filter as an example, the concrete steps of even extending the image expansion process are illustrated below [45]. The principle of specific image expansion is shown in Figure 4.

(1) Because the radius of the neighborhood window of the filter template window size is 2, an extended image of 7×7 image size is generated.
(2) Fill the gray value of the original image pixel into the corresponding position coordinate of the extended image
(3) Fill the left and right boundaries. As shown in Figure 4, the values of the pixels P_1, P_4, and P_7 are first copied and filled as the values on the left side of the extended image closest to the left boundary of the original image. Then copy the values of the pixels P_2, P_5, and P_8 as the values on the left boundary of the extended image. The filling process of the data on the right side of the extended image is similar to that on the left part.
(4) Fill the upper and lower boundaries. Unlike the left and right boundary fills, you need to consider the fill of the four corners. Therefore, you need to copy the pixel value of the entire row to the extended location. As shown in Figure 4, the values of P_2, P_1, P_1, P_2, P_3, P_3, and P_2 line pixels are first duplicated as the values closest to the upper boundary of the original image in the extended image. The values of P_5, P_4, P_4, P_5, P_6, P_6, and P_5 line pixels are then duplicated to fill in the values of the upper boundary of the extended image. The filling process of the lower part of the extended image is similar to that of the upper part.

C. ALGORITHM HOT SPOT ANALYSIS

An experimental analysis of the image weighted mean filtering algorithm is carried out. The experimental data are test image with an image size of 5326×5764, and the 3×3 weighted mean filter template shown in Figure 3. The compiling and running environment of the software is as follows: the operating system is Windows 10 64 bits, the compiler is Microsoft Visual Studio 2015, and the CPU is Intel Core i7 6700@3.4 GHz. The running time distribution of each functional module of the algorithm is shown in Table 1.

TABLE 1. Running time distribution of each functional module of the weighted mean filtering algorithm.

Main calculation steps	Execution time /ms	Time percentage / %
Extended image	739.00	8
Local image processing	6003.00	65
Other parts	2493.00	27
Total	9235.00	100

The percentage of running time for each functional module is shown in Table 1. It can be seen that the operation time of the image local processing function module is the longest, accounting for 65% of the total running time of the algorithm, the running time of the extended image function module accounts for 8%, and the running time of other parts accounts for 27%. It can be seen that the local image processing is a hot module of the image weighted mean filtering algorithm, and the module is computationally intensive. Therefore, if we analyze the parallelism of this hot module, choose to parallelize the module, and then transplant the module to GPU for calculation, we can get a good acceleration effect.

D. PARALLELIZATION ANALYSIS OF ALGORITHM

The data of an image in a computer is expressed as a real two-dimensional matrix, and the storage mode in memory can transform the two-dimensional structure into a one-dimensional linear structure in the order of rows. The pixels in an image are scattered and independent of each other. The image mean filtering algorithm uses the 3×3 weighted mean filter template to calculate the convolution of each pixel in the image, the masking coefficient and the corresponding pixel gray value of the mask cover area are dotted multiplication and summed up. Although the pixel is correlated with the remaining pixels in the mask coverage area, the whole convolution calculation process only reads the original image and does not change the original image data. Therefore, the convolution result contributes to only one pixel but has no effect on the other pixels, and the calculation of one pixel will not affect the calculation results of the other pixels. In the algorithm, the calculation of each pixel can be carried out at the same time, with parallelism. That is, the weighted mean filtering algorithms directly called by different pixels are independent and unrelated to each other. This fully proves that it is feasible to convert the serial algorithm into the parallel algorithm. In this paper, the weighted mean filtering convolution of different pixels can be assigned to different work-items without having to worry about their dependence on each other. Therefore, in the parallel computing based on OpenCL architecture, the local image processing functions of different pixels can be processed at the same time, because their calculations are interdependent.

IV. OpenCL IMPLEMENTATION OF THE IMAGE MEAN FILTERING ALGORITHM

A. PARALLEL ALGORITHM DESCRIPTION

According to the characteristics of OpenCL architecture, each work-item in Single Instruction Multiple Thread (SIMT) model is only responsible for the mean filtering of one pixel and then stores the processing results to the corresponding pixel location. The formal description of the image weighted mean filtering parallel algorithm is shown in listing 1.

```
Algorithm  Parallel algorithm of image weighted mean filtering based
           on SIMT model

Input:   Image pixel matrix $srcImageData$ with image size $M \times N$,
         Mean filter template size $MODELDIM \times MODELDIM$
Output: Pixel matrix $desImageData$ after image mean filtering
Begin
    $In\ CPU$ :
    Input image array $srcImageData$
    $srcImageData' \leftarrow$ even extension of image

    $In\ GPU$ :
    Weighted mean filter templates are stored in constant memory
    for $i = 0$ to $(M \times N - 1)$ par - do
        for $j = 0$ to $(MODELDIM \times MODELDIM - 1)$
            Weighted average of neighborhood gray value of current pixel
             and image filtering window coefficient
        end for
    end for

    $In\ CPU$ :
    Output image mean filtering result array $desImageData$ and display
    filtered image
End
```

LISTING 1. The formal description of the weighted mean filtering parallel algorithm.

(1) Assuming that the image size is $M \times N$, if implemented by a single-thread CPU, the image weighted mean filtering serial algorithm is calculated by traversing the whole image pixel data, so the time complexity is $O\left(M \times N \times MODELDIM^2\right)$.

(2) When using a multi-core CPU for parallel computing, the number of threads is created based on the number of CPU cores *csum*. Each thread is assigned to the image block in a self-heuristic way, and the convolution operation of the weighted mean filtering of the pixels in each image block is carried out in parallel. Therefore, the time complexity of the algorithm is reduced to $O\left(\left(M \times N \times MODELDIM^2\right)/csum\right)$.

(3) The multi work-items of many-core GPU are used to calculate the image local processing function in parallel. A work-item is responsible for calculating the weighted mean filtering convolution of a pixel, and the time complexity of the algorithm is reduced to $O(MODELDIM)^2$. If all the pixels in the image are not processed in a kernel function, each work-item will execute the kernel function that completes the image local processing function at least $(M \times N)/tsum$ times, where *tsum* represents the total amount of active work-items. At this point, the time complexity will be $O\left(\left(M \times N \times MODELDIM^2\right)tsum\right)$.

It should be noted that, in general, the core number *csum* in a multi-core CPU is not large, and the Intel Core i7 6700 (four cores) is used in this paper. On the other hand, there are a large number of active work-items that can be maintained in GPU, that is, *tsum* is always a large value. Therefore, $csum \ll tsum$. Therefore, there are three image weighted mean filtering algorithms with time complexity relation

$$\begin{aligned} &O\left(\left(M \times N \times MODELDIM^2\right)/tsum\right) \\ &\ll O\left(\left(M \times N \times MODELDIM^2\right)/csum\right) \\ &< O\left(M \times N \times MODELDIM^2\right). \end{aligned}$$

B. CALCULATION PROCESS

The GPU parallelization process of image mean filtering is shown in Figure 5.

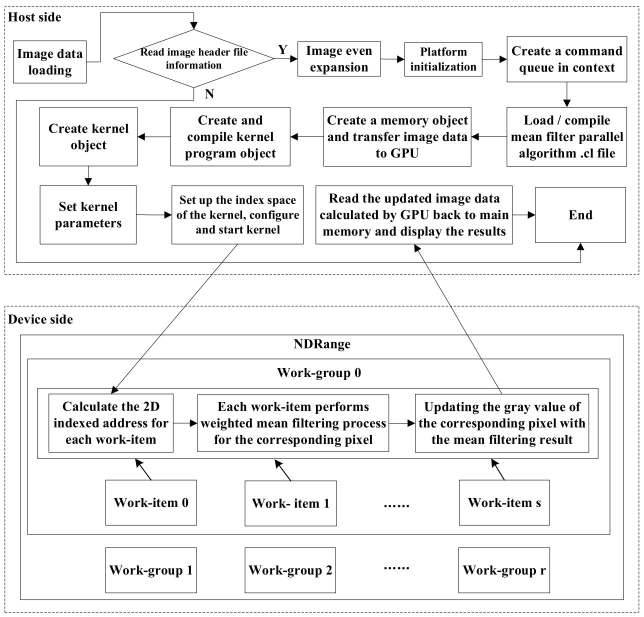

Host side
Image data loading
Read image header file information
Y
N
Image even expansion
Platform initialization
Create a command queue in context
Create kernel object
Create and compile kernel program object
Create a memory object and transfer image data to GPU
Load / compile mean filter parallel algorithm .cl file
Set kernel parameters
Set up the index space of the kernel, configure and start kernel
Read the updated image data calculated by GPU back to main memory and display the results
End
Device side
NDRange
Work-group 0
Calculate the 2D indexed address for each work-item
Each work-item performs weighted mean filtering process for the corresponding pixel
Updating the gray value of the corresponding pixel with the mean filtering result
Work-item 0
Work- item 1
......
Work-item s
Work-group 1
Work-group 2
......
Work-group r

FIGURE 5. Parallel computing process of image mean filtering.

The main steps to realize the parallel computation of the weighted image mean filtering are as follows:

(1) Create the platform object, get the OpenCL platform, and create the GPU device. Establish the OpenCL platform context.

(2) Creates a command sequence object from the context and the specified device, and uses the command queue to establish a connection between the device and the context.

(3) The image data are read in, the parameters are calculated and the data are transmitted to the video memory.

Get the identification, height, width, color, size, and other data of the image.

According to the size of the image, the storage area is established on the device. CPU reads the image data into the host memory and calls the OpenCL function to transfer the data from the host memory to the device global memory. Due to the limitation of bandwidth, the data communication between host memory and video memory has become the biggest bottleneck restricting the speed improvement of the system. Therefore, the whole data transmission is used in the parallel algorithm instead of multiple block transmission.

(4) After the system first loads and compiles the image mean filter kernel to generate the executable file, when it runs again, it uses the precompiled offline to generate the binary executable file and compiles the executable system on the specified device. Create and compile program objects.

(5) Create a kernel object from a program object.

(6) Set the passed parameters for the kernel object to start the kernel. The system schedules kernel functions to perform image mean filtering calculations. This phase consists of the following steps:

Determine the execution configuration of the kernel. According to the data density and computation, the index space dimension and work-group dimension of the device are set.

The decomposition of the input data. The work-items in each work-group determine the pixel objects that need to be calculated based on the index address.

Kernel functions are placed in the command queue through clEnqueueNDRangeKernel to perform parallel computing between work-items.

The weighted mean filtering results of all the pixels of the image calculated by GPU are all written to the global memory.

(7) The data is read out and the image output is carried out. The calculation results are transmitted from the video memory back to the host memory, and the image mean filtering results are displayed.

C. PARALLEL COMPUTING METHOD

1) DESIGN OF THE KERNEL FUNCTION

In order to carry out a large number of data processing, the OpenCL kernel needs to carry out a large number of data partition operations. The more reasonable the data partition processing is, the shorter the data processing time is. The smallest unit that can be executed independently in OpenCL is a work-item. Several work-items form a work-group. Work-items within a work-group can share resources within the work-group and execute concurrently within the work-group. Different work-groups execute in parallel on different compute units.

OpenCL uses the clEnqueueNDRangeKernel function to put specific tasks into the queue for execution and divides the tasks by setting parameters. Different work-items complete different tasks for different data and work-items are executed in parallel. OpenCL can use a data-parallel programming model to construct a weighted mean filter in parallel. The NDRange parameter of clEnqueueNDRangeKernel is an array, and the dimension of the index space is set to two dimensions according to the characteristics of the digital image. As shown in Figure 6, NDRange is a two-dimensional array, and one element of the array is a work-group, each of which can independently carry out weighted mean filter processing for the corresponding sub-image blocks. A work-group contains a fixed number of work-items, each of which can independently carry out weighted mean filter processing for the corresponding for pixels. In this paper, the dimension of the work-group is set to (S_x, S_y), and the whole image is logically divided into $((M + S_x - 1)/S_x \times (N + S_y - 1)/S_y)$ sub-image blocks, that is, in theory, these sub-image blocks can be put into compute units for parallel computing by the same number of work-groups at the same time. Therefore, the OpenCL workspace NDRange is configured with $(M + S_x - 1)\big/S_x$ work-groups in the X direction and $\big(N + S_y - 1\big)\big/S_y$ work-groups in the Y direction. Because different GPU models have different performance, the maximum number of work-items supported varies. To take advantage of the best performance of GPU, you need to find the best number of work-items to achieve the best performance of GPU. Through the experiment, 256 work-items are enabled to execute kernel functions per work-group.

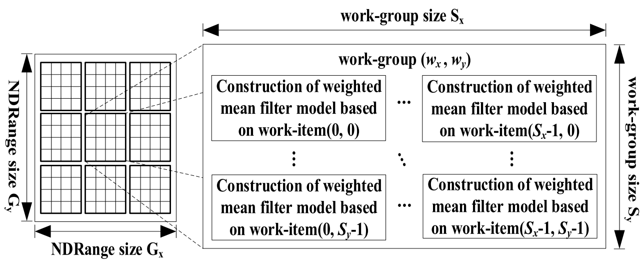

work-group size S_x
work-group (w_x, w_y)
Construction of weighted mean filter model based on work-item(0, 0)
Construction of weighted mean filter model based on work-item(S_x-1, 0)
Construction of weighted mean filter model based on work-item(0, S_y-1)
Construction of weighted mean filter model based on work-item(S_x-1, S_y-1)
NDRange size G_y
NDRange size G_x
work-group size S_y

FIGURE 6. The principle of parallel constructing weighted mean filter model using OpenCL.

2) CONVOLUTION CALCULATION

Weighted mean filtering is realized by discrete sliding window convolution calculation. Figure 7 shows a schematic diagram of a convolution calculation. In a two-dimensional convolution calculation, if the convolution of $p[i][j]$ the pixel in the range of the original image is calculated, the weighted mean filter template and the two-dimensional sub-matrix with the same size as the template matrix centered on $p[i'][j']$ in the extended image matrix $srcImageData'$ are required. In the convolution calculation, it is necessary to multiply and accumulate the data of the template matrix and the corresponding submatrix in the extended image matrix and divide by the weighted sum to obtain the updated results of the original image after the weighted mean filtering. For all the pixels, the parallel convolution calculation process is carried out, and the parallel processing of the weighted mean filtering algorithm is realized.

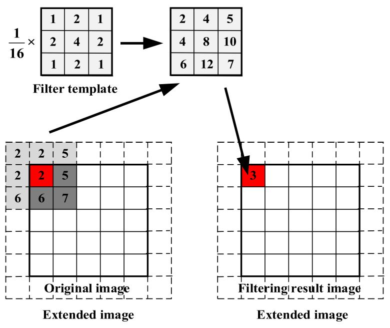


FIGURE 7. Two-dimensional convolution calculation.

TABLE 2. The use effect of constant memory.

Image size	Run time for unused constant memory/ms	Run time for using constant memory/ms	Acceleratio n ratio	p
1354×1675	19.35	15.61	1.24	23.96
2481×2768	39.63	29.35	1.35	35.03
4352×4877	379.85	271.31	1.40	40.01
5326×5764	964.98	545.18	1.77	77.00
6248×6792	2408.04	1594.72	1.51	51.00

D. OPTIMIZATION STRATEGY

1) CONSTANT MEMORY OPTIMIZATION

The constant memory in GPU is read-only. If each work-item in half-warp reads data from the same address in constant memory, GPU only produces one read request and then broadcasts the data to each work-item. At the same time, because the content of the constant memory will not change and the constant memory has a cache mechanism, GPU can put the constant data directly into the L1 cache. After the first read from an address in constant memory, when other half-warp requests the same address, the cache is hit, speeding up access. Therefore, reading the same data from the constant memory can save a large amount of memory bandwidth compared with reading the data from global memory.

Because in the process of image weighted mean filtering, the coefficients of the weighted mean filtering template will not change, and the filter template is used in the processing of each work-item. In order to improve the running efficiency, the parallel algorithm copies the filter template data directly from CPU memory to GPU constant memory. In this way, each work-item can accelerate access to the mean filter and save the running time of the algorithm. In order to represent the degree of improvement in the performance efficiency of the weighted mean filtering algorithm before and after using constant memory, the performance improvement is defined as shown in formula (2).

$$p = \frac{\left(T_{bef} - T_{aft}\right)}{T_{aft}} \times 100\% \tag{2}$$

Among them, p is the performance improvement ratio, T_{bef} is the execution time of the algorithm before performance optimization, and T_{aft} is the execution time of the algorithm after performance optimization. The system effect of the image weighted mean filtering algorithm before and after using constant memory is compared as shown in Table 2.

Table 2 shows that the running time of the algorithm is different before and after the weighted mean filtering template is stored in constant memory. The global memory is used to lag the calculation time before optimization. After optimization, the running time of the algorithm is shortened, the computing time delay is reduced, the maximum acceleration ratio of 1.77 times is obtained, and the execution efficiency of the algorithm is improved by 77%.

2) LOCAL MEMORY OPTIMIZATION

In OCL_MF, the source image data is simply copied to the global memory on the GPU, which gains great benefits. Any work-item in the entire workspace can read/write anywhere in the global storage, and the global memory can be accessed either from the GPU side or from the CPU side. It takes about 400 ∼ 800 clock cycles to access the global memory, and a large number of work-items repeatedly access the source image data from the global memory, resulting in a high delay, which will lead to idle GPU computing resources. The local memory is located in the GPU chip and is much faster than the global memory. In the case of no bank conflict, the latency of the local memory is almost only 1/100 of that of the global memory, and the access speed is as fast as that of the registers.

In the convolution calculation of the image local processing function module, when the source image data is convoluted with the filter coefficients, each image data is reused for $1 \sim MODELDIM \times MODELDIM$ times, resulting in a large overhead of global storage access. Therefore, the local memory can be used instead of the global memory to put the source image block data into the local memory. Split the source image into $\left((M+S_x-1)\big/S_x \times \left(N+S_y-1\right)\big/S_y\right)$ sub-image blocks and assign $\left((M+S_x-1)\big/S_x \times \left(N+S_y-1\right)\big/S_y\right)$ work-groups. Each work-group is allocated local memory space, each sub-image block data is stored in the local storage space of the corresponding work-group, and S_x, S_y work-items are allocated in the X, Y direction of the work-group respectively, and each work-item completes a convolution calculation.

3) OPTIMIZE WORK-ITEM CONFIGURATION

The layout of different work-groups have different performance for memory access. The number of work-items per work-group should be an integer multiple of warp size. When there are enough work-groups, the waste of computing resources caused by insufficient warp block filling is avoided. Based on the requirements of global memory merge access and partition conflicts, it is recommended that each work-group use 128 to 256 work-items. This improves the efficiency of accessing global memory. As many work-items as possible are executed at the same time, making it easier to hide memory delays. The work-group should try to make the dimensions in the X and Y directions multiple of the warp size. Table 3 shows the system operation time when setting up a different number of work-items in a work-group when the image size is 2481×2768. It can be seen that the number of work-items in each work-group varies with the corresponding operation time. When the work-group is configured to be 16×16, the system performance is optimal. Assigning more work-items to each work-group can achieve effective time segmentation, thereby increasing the speed of operation. However, if there are too many work-items in each work-group, the fewer registers available for each work-item, and the fewer work-groups that can actually be scheduled to run on the compute units and even cause the kernel to fail to start due to insufficient registers.

TABLE 3. Effect of work-group size on operation speed.

Number of work-items in work-group	Run time/ms
4×4	44.07
8×8	32.87
16×16	29.35
32×32	36.77

4) DATA TRANSMISSION OPTIMIZATION

By using pinned host memory for data transmission optimization, the overhead of data transmission caused by communication between devices in the system is reduced. The extended image and filtered image of the OCL_MF parallel algorithm on the host are generally stored in the pageable host memory, and the GPU device can not directly access the pageable host memory. As shown in Figure 8 (a). Before the data transmission between the host and the device, it is necessary to open up a temporary pinned host memory, to copy the expanded image and filtered image data from the paged host memory to the pinned host memory, and finally transfer the image data directly from the pinned host memory to the global memory of the device. As shown in Figure 8 (b).

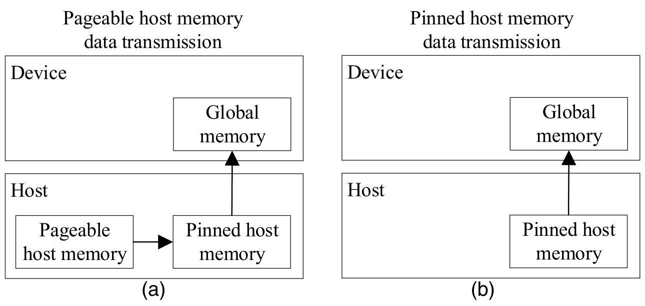


FIGURE 8. Data transmission.

The zero-copy memory is actually a pinned host memory on a host. By using zero-copy memory, the copying process from pageable host memory to pinned host memory is avoided. Unified memory creates a pool of managed storage shared by CPU and GPU to bridge the gap between CPU and GPU. Both CPU and GPU can access managed storage using a single pointer, and the key is that the system automatically migrates the data allocated in the unified memory between the host and the device. Unified Memory combines the advantages of explicit copy and zero-copy access: GPU can access any page of the entire system memory, while migrating extended image and filtered image data to its memory on demand to achieve high-bandwidth access.

E. OTHER PARALLELIZATION SCHEMES

In order to facilitate the performance comparison of various parallelization schemes, this paper implements a parallel algorithm that meets the weighted mean filtering algorithm principle of Section III based on the OpenMP and CUDA parallel computing architectures, respectively.

1) THE WEIGHTED MEAN FILTERING PARALLEL ALGORITHM BASED ON OPENMP

In the Windows system, use the built-in Microsoft Visual Studio 2015 compiler of OpenMP, set 'C/C++ → Language → OpenMP Support' to Yes (/openmp) in the property page of Microsoft.Cpp.x64.user of the project, and open the OpenMP compilation option in Visual Studio 2015. Then change the option 'C/C++ → Gode Generation → Runtime Library' to Multi-threaded Debug (/ MTd).

OpenMP is a coarse-grained parallel. According to the number of CPU cores, sub-threads are enabled through omp_set_num_threads (number of threads) to execute parallel blocks in parallel. When the weighted mean filtering parallel computing system based on OpenMP is executed by the main thread, the #pragma omp parallel for compilation guidance instruction before the function of local image mean filtering processing is encountered, the following loop bodies are decomposed by multithreading. Each thread processes one pixel of the original image in parallel and ends with the main thread. This makes rational use of system resources and improves computational efficiency.

OpenMP allocates a total of *csum* sub-threads, which need to deal with the computing tasks of $M \times N$ pixels in image size. In general, the number of sub-threads is much less than the number of pixels. Therefore, each sub-thread is responsible for the calculation of multiple pixels. The basic idea of partition is to decompose the pixels into *csum* groups according to rows, each of which is calculated by a sub-thread. The key point is to establish the mapping relationship between the thread index and the pixel point index. The task decomposition of each sub-thread is completed through a double loop below:

```
int threadIdx = omp_get_thread_num(); // Gets the current sub-thread index
for (int i = threadIdx; i < M; i + = csum)
    for (int j = 0; j < N; j++)
        PixelNodeCompute(...);
```

That is, for the current thread *threadIdx* in the parallel domain, the thread is responsible for processing all the calculations of the pixel points of each column in the image from the 0th column of the row $i = threadIdx$ to the range of column $(N - 1)$, where the increment interval of the row *csum* is the total number of sub-threads *csum*. The reason is that *csum* sub-threads have been allocated, and the previous $0 \sim (csum - 1)$ lines have been assigned to these *csum* sub-threads for parallel processing. Therefore, the previous *csum* lines do not need to be recalculated, which is the key to the partition of thread parallel tasks. $i = i + csum$ means to continue searching in the X direction for rows that need to be calculated by the thread to ensure that the calculation is not missed. $i < M$ guarantees that the index access to the pixels will not exceed the range of the image pixel index (not beyond the boundary). In the process of image mean filtering, the number of threads is not the more the better. In order to select the optimal number of threads, when the image size is 2481 × 2768, 1, 2, 3, 4, and 8 threads are designated in the experiment to execute the image weighted mean filtering function respectively, and the results are shown in Table 4.

TABLE 4. The influence of thread quantity on operation speed.

Number of threads	Running time/ms
1	519.00
2	446.63
3	331.34
4	220.85
8	330.74

It can be seen from Table 4 that when the number of CPU threads is set to 4, the operation time of the OMP_MF parallel algorithm is the shortest. When the number of CPU threads is set to 8, the computing time of the OMP_MF parallel algorithm increases. It can be seen that the execution time of the OMP_MF parallel algorithm optimized by OpenMP is reduced to about 43% of the original, and it can be seen that the more threads are used, the higher the efficiency. When the number of threads exceeds 4, the goal of further improving efficiency can no longer be achieved simply by increasing the number of threads. According to the above analysis, this paper uses four threads to execute the algorithm.

2) THE WEIGHTED MEAN FILTERING PARALLEL ALGORITHM BASED ON CUDA

CUDA is a fine-grained parallel. The parallelizable convolution calculation and mean calculation for each pixel of the weighted mean filtering algorithm are divided into one task. The task is handled by the corresponding global functions. During the running of the algorithm, the global function corresponds one-to-one with the grid, and the grid assigns the task to the blocks, and the block reassigns the task to the threads for processing. The global function is invoked by the host side, and parallel computation is performed in multiple computation and processing units at the GPU side to achieve optimization and acceleration.

V. EXPERIMENTAL TEST DATA AND EXPERIMENTAL RESULTS DISCUSSION

The test results of the weighted mean filtering serial / parallel algorithms described in this article will be introduced in this section. The core source code of the weighted mean filtering parallel algorithm is developed by OpenCL C language. The parallel algorithm consists of two parts, one is the main program executed on the host machine, and the other is the kernel program executed on the OpenCL device. The kernel codes implement the convolution calculation and mean calculation.

A. EXPERIMENTAL COMPUTING PLATFORMS

Our experimental work is carried out on the CPU/GPU heterogeneous hybrid parallel computing platform. The hardware and software environment of the experiment is configured as follows. The hardware environment parameters are shown in Table 5.

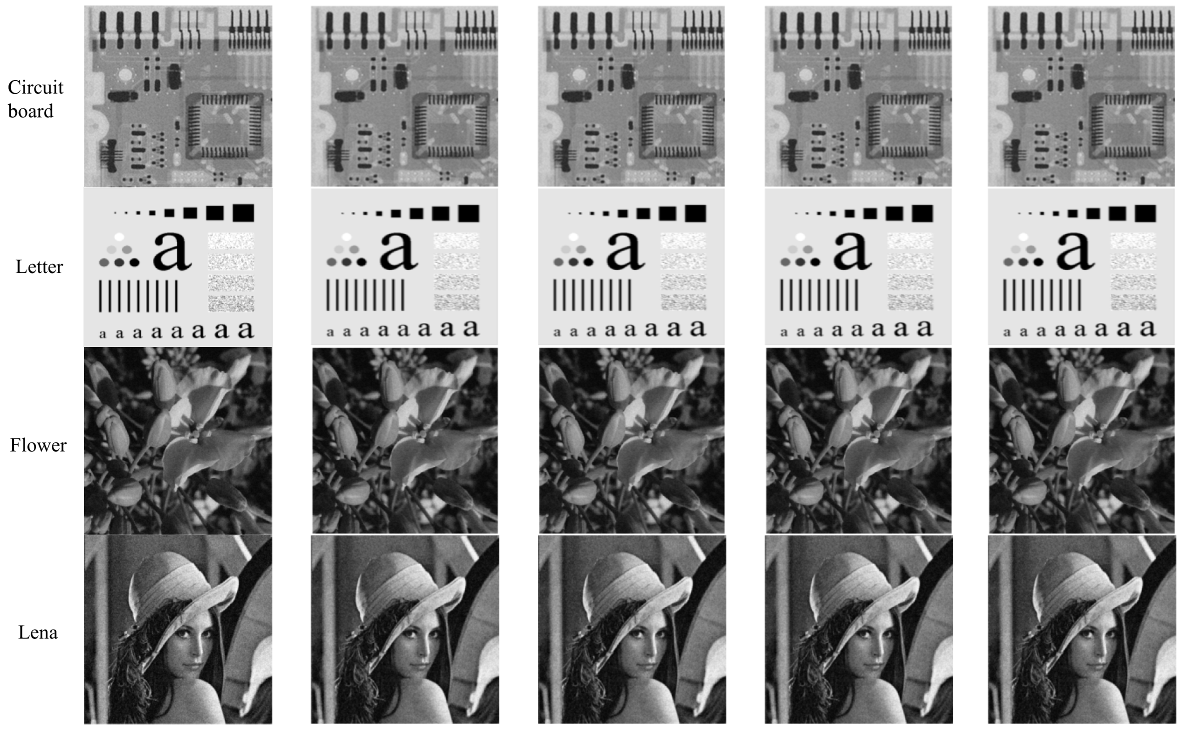
TABLE 5. The main performance parameters.

Performance parameters \ Platform	Platform 1	Platform 2
CPU	Intel Core i7 6700 3.4 GHz (four cores)	Intel Core i7 6700 3.4 GHz (four cores)
System memory	16.0 GB	16.0 GB
GPU	NVIDIA GeForce GTX 1080	AMD Radeon RX 5700
GPU cores	2560	2304
GPU core frequency	1607 MHz	1465 MHz
Video memory	8 GB GDDR5	8 GB GDDR5
Video memory bandwidth	320 GB/s	448 GB/s
Video memory bit width	256 bits	256 bits
Single-precision floating-point computing capacity	10.6 TFLOP/s	7.95 TFLOP/s

The operating system of the host software environment is Microsoft Window 10 64-bits, the CPU simulation experiment software is MATLAB R2018b, the NVIDIA GPU application programming interface (API) is CUDA Toolkit 10.0, the computing development kit of AMD GPU is AMD APP SDK 2.7, OpenCL 1.2 is supported internally in the system, multicore processor support environment OpenMP 3.0, and the development environment is Microsoft Visual Studio 2015. The version of the NVIDIA's driver is 411.31 and the version of the AMD's driver is 8.801.0.0.

B. EXPERIMENTAL RESULTS

In order to carry out the comparative experiment of multiple groups of data, the original image data need to be preprocessed. Eight groups of experimental data with image sizes of 256 × 256, 345 × 758,1354 × 1675, 2481 × 2768,



(a) Original noise image (b) CPU_MF effect show (c) OMP_MF effect show (d) CUDA_MF effect show (e) OCL_MF effect show

FIGURE 9. **Gaussian noise image processing effect under different computing platforms.**

4352 × 4877, 5326 × 5764, 6248 × 6792, and 8134 × 8256 were obtained by clipping.

In order to verify the effectiveness of the algorithm, four different categories of images of circuit boards, letters, flowers, and Lena figures are selected in the experiment. Figure 9 (a) noise image is obtained by adding a mean value of 0 and a variance of 0.005 Gaussian noise to the four original images by MATLAB software. Figure 9 (b), (c), (d), and (e) are the images processed by the CPU_MF, OMP_MF, CUDA_MF, and OCL_MF systems, respectively.

The contrast experiment of image weighted mean filtering was carried out with eight images of different sizes preprocessed. The CPU_MF, the OMP_MF, the CUDA_MF, the OCL_MF based on AMD GPU, and the OCL_MF based on NVIDIA GPU are used to run respectively, and the processing time is recorded, as shown in Table 6.

The verification of the efficiency of parallel algorithms under various frameworks can intuitively use the acceleration ratio as a measure of acceleration effect, which is defined as follows:

The speedup refers to the ratio of the operation time of the CPU_MF serial algorithm to the operation time of the parallel algorithm.

The relative speedup 1 refers to the ratio of the operation time of the OMP_MF parallel algorithm to the operation time of the OCL_MF parallel algorithm based on NVIDIA GPU.

The relative speedup 2 refers to the ratio of the operation time of the CUDA_MF parallel algorithm to the operation time of the OCL_MF parallel algorithm based on NVIDIA GPU.

TABLE 6. **Execution time of weighted image mean filtering algorithm under different computing platforms.**

Image size / px	Serial processin g time/ms	Parallel processing time/ms			
		OpenMP	CUDA	OpenCL (AMD)	OpenCL (NVIDIA)
256×256	3.00	2.91	0.62	0.60	0.58
345×758	49.00	45.37	2.54	2.48	2.45
1354×1675	269.00	199.26	13.89	13.62	13.51
2481×2768	519.00	220.85	26.68	25.58	24.86
4352×4877	4577.00	1594.77	280.52	265.41	263.64
5326×5764	9235.00	2841.54	675.24	572.18	536.57
6248×6792	24479.00	7285.42	1837.86	1606.38	1581.35
8134×8256	31208.00	8815.82	2393.46	2072.67	2032.86

The speedup reflects the overall improvement of the efficiency of the parallel algorithm compared with the CPU_MF serial algorithm under the corresponding parallel computing architecture and can be used to objectively evaluate the speed of the actual system. The relative speedup 1 reflects the improvement of the efficiency of the OCL_MF parallel algorithm based on NVIDIA GPU compared with the OMP_MF parallel algorithm. Relative speedup 2 reflects the improvement of the efficiency of the OCL_MF parallel algorithm based on NVIDIA GPU compared with the CUDA_MF parallel algorithm. As shown in Table 7.

TABLE 7. Performance comparison of weighted image mean filtering parallel algorithm under different computing platforms.

Image size / px	Acceleration ratio				Relative acceleration ratio 1	Relative acceleration ratio 2
	OpenMP	CUDA	OpenCL (AMD)	OpenCL (NVIDIA)		
256×256	1.03	4.84	5.00	5.17	5.02	1.07
345×758	1.08	19.29	19.76	20.00	18.52	1.04
1354×1675	1.35	19.37	19.75	19.91	14.75	1.03
2481×2768	2.35	19.45	20.29	20.88	8.88	1.07
4352×4877	2.87	16.32	17.25	17.36	6.05	1.06
5326×5764	3.25	13.68	16.14	17.21	5.30	1.26
6248×6792	3.36	13.32	15.24	15.48	4.61	1.16
8134×8256	3.54	13.04	15.06	15.35	4.34	1.18

C. VALIDITY VERIFICATION

The image weighted mean filtering algorithm implemented in CPU serial calculation is used as the benchmark CPU program for GPU transplantation and optimization. All parameters of the parallel algorithm system are consistent with the benchmark program. Therefore, the validity verification is only compared with the running results of the CPU serial algorithm.

1) CONSISTENCY OF RESULTS AT THE MACRO-LEVEL

From the experimental results of Figure 9 (b) — (e), it can be seen that in the experiment, the original image is polluted by additive Gaussian noise, and the noise is attenuated by serial and parallel processing of the weighted mean filtering algorithm. Smoothed local changes in the original image. The running results of the image weighted mean filtering serial system and the parallel system are the same, and there is no identifiable difference with the naked eye.

2) CONSISTENCY OF RESULTS AT THE MICRO-LEVEL

Figure 10 shows the comparison of the histograms of the image before and after the weighted mean filtering process. From the analysis results of Figure 10, it can be seen that the serial processing of the weighted mean filtering algorithm in the experiment is the same as the corresponding data of the image histogram of all kinds of parallel processing, that is, the number of pixels with the same gray level is the same, and the consistency of the processing results is maintained.

D. EXPERIMENTAL DATA ANALYSIS

1) SYSTEM BOTTLENECK ANALYSIS

It requires $M \times N$ times the memory read operations of the extended image data and $M \times N$ times the memory write operations of the mean filter image processing result. The size of the image is $M \times N = 5326 \times 5764$, and each pixel takes up storage space of 2B. As a result, the total amount of image data accessed by the memory is approximately 0.12 GB. Divided by the elapsed time of 0.00042 s for the kernel execution on the device. The actual bandwidth obtained by the system is approximately 285.71 GB/s, which is close to the theoretical bandwidth of GeForce GTX1080 display memory. Thus, it can be seen that the bandwidth of the global memory limits the further improvement of the efficiency of the mean filtering parallel algorithm based on OpenCL architecture. Therefore, the performance bottleneck for the mean filtering parallel algorithm based on OpenCL architecture is the display memory bandwidth.

From Table 7, it can be seen that the acceleration of the algorithm based on the CPU + GPU is obvious, but the acceleration ratio of the GPU parallel algorithm shows a slow decreasing trend with the increase of the image size. This is because in the design of OpenCL parallel algorithm, the task of reading and outputting image data is assigned to CPU, but the performance of this process is not improved. With the increase of image size, the time-consuming of reading and outputting image data also increases. Comprehensive analysis shows that the bandwidth of video memory and the data transmission bandwidth between main memory and video memory become the performance bottlenecks of OpenCL accelerated the mean filtering parallel algorithm.

2) ANALYSIS OF OPERATION TIME OF WEIGHTED MEAN FILTERING ALGORITHM UNDER DIFFERENT ARCHITECTURE

The experimental images with different image sizes are processed by weighted mean filtering on four computing platforms, and the serial running time of CPU is compared with the running time under three different parallel computing frameworks, as shown in Figure 11. When the image size is the same, the execution time on different parallel computing platforms is reduced to varying degrees compared with the CPU serial execution time, that is, the acceleration effect is obtained. For example, for the weighted mean filter with an image size of 2481×2768, the serial operation time is 519.00 ms, and the operation time is shortened to 220.85 ms in the OpenMP computing platform. The computing time on the parallel computing platform based on CUDA architecture is greatly reduced to 26.68 ms, while the low computing time on the OpenCL-based parallel computing platform is about 24.86 ms.

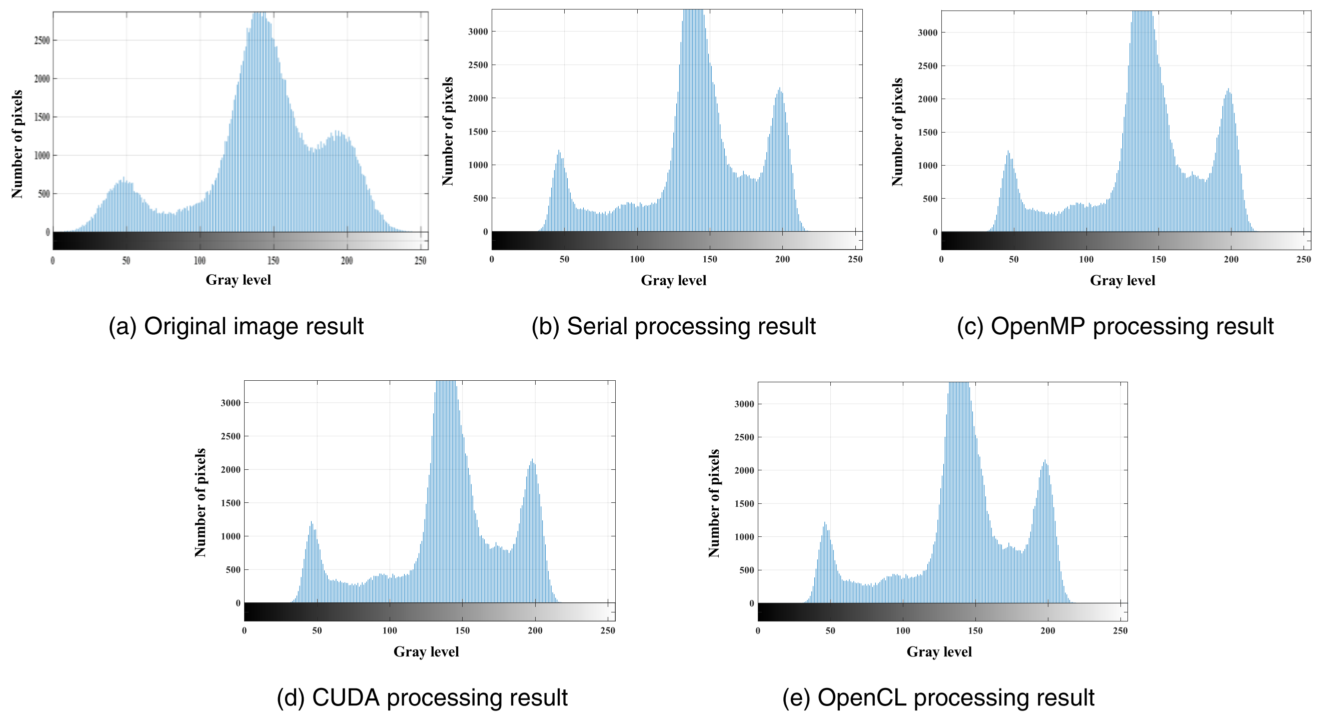


FIGURE 10. Comparison of histograms before and after image weighted mean filtering.

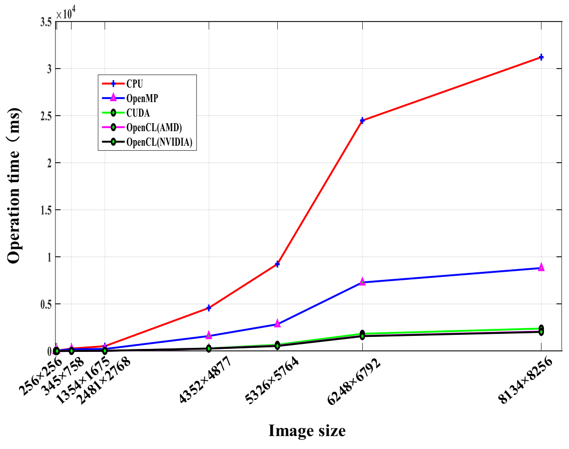


FIGURE 11. Comparison of operation time of the weighted mean filtering algorithm.

Under the same image size, the running time of the weighted mean filtering algorithm after OpenMP parallelization is significantly less than that of the traditional serial algorithm. And under the same number of threads, with the increase of the image size, the running time is getting longer and longer, which is basically in line with the trend of linear growth.

According to the above analysis, because the data interaction between host memory and GPU memory requires a certain amount of time overhead, when the image size is small, this part of the overhead has little impact on the final computing time. When the scale of the image data is very large, the proportion of data interaction time has increased, and the GPU computing time is not enough to cover the time overhead caused by transmission delay, so the effect of OpenCL acceleration tends to slow down.

The overall acceleration effect of the algorithm in this paper is compared with that of reference [30]. As most of the other literatures use the weighted mean filtering algorithm for various application researches, there is little research on the acceleration effect of the weighted mean filtering algorithm. Therefore, it is impossible to compare the acceleration effect directly. According to the test data provided in reference [30], when the image size is 256×256, the operation time of the mean filtering parallel algorithm based on CUDA acceleration in reference [30] is 1.11 ms, and the speedup is 3.91 times. According to the test results of Table 6 and Table 7, we can see that the operation time of the weighted mean filtering parallel algorithm based on OpenCL acceleration in this paper is 0.58 ms, and the speedup is 5.17 times. Therefore, the operation time of the parallel algorithm in this paper is shorter than that in reference [30], and better acceleration performance is achieved.

3) COMPARATIVE ANALYSIS OF THE ACCELERATION EFFECT OF WEIGHTED MEAN FILTERING ALGORITHM IN PARALLEL COMPUTING ARCHITECTURE

As can be seen from Figure 12, the image average filtering algorithm based on multi-core CPU has achieved certain acceleration results. Compared with the serial algorithm, with the increase of the image size, the overall acceleration is from 2 times to nearly 4 times. Due to the restriction of the number of CPU cores, it is difficult for the OpenMP parallel algorithm

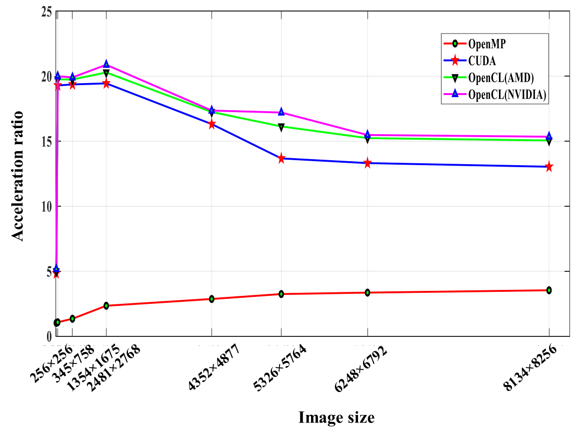


FIGURE 12. Acceleration ratio trend diagram of the weighted mean filtering parallel algorithm.

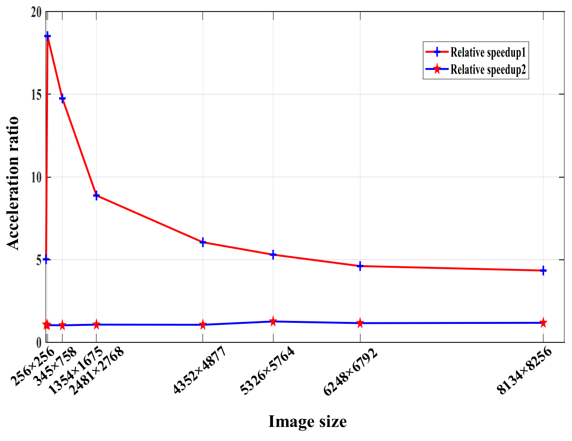


FIGURE 13. Relative acceleration ratio trend graph.

to achieve a high acceleration ratio. However, the acceleration effect of the GPU parallel algorithm is very obvious. For example, when the image size is 2481 × 2768, the system achieves 19.45 times speedup in the parallel algorithm accelerated by CUDA. On the other hand, on the NVIDIA platform accelerated by OpenCL, the parallel algorithm is 20.88 times faster than the serial algorithm, which greatly saves the noise reduction time. However, with the further increase of the image size, the OpenCL speedup shows a slow downward trend. The reason for this phenomenon is that the data transmission bandwidth between the host and the device is much lower than that between the device and the device, so the data interaction between host memory and GPU memory has a certain time overhead. When the image scale is large, the proportion of data interaction time increases gradually, the GPU parallel computing time is not enough to cover the system transmission delay overhead, the data transmission overhead has a certain impact on the computing time of the system. Even so, the performance of the parallel algorithm is improved 15.35 times when the image size is 8134 × 8256. Therefore, the weighted mean filtering parallel algorithm based on the OpenCL acceleration proposed in this paper can meet the real-time requirements.

Figure 13 shows that the performance of the optimized OpenCL parallel algorithm is slightly faster than that of the CUDA parallel algorithm, and the relative speedup 2 shows a maximum improvement of 1.26 times. At the same time, it reflects that with the increase of image size, the performance gap between the two parallel algorithms tends to widen. Therefore, the performance of the optimized OpenCL parallel algorithm is close to that of the CUDA parallel algorithm. When the image size is large, the OpenCL parallel algorithm has more performance advantages. The relative speedup 1 shows that the performance of the weighted mean filter parallel algorithm based on OpenCL is much better than that based on the OpenMP platform, and the maximum speedup is 18.52 times.

4) PORTABILITY ANALYSIS OF OPENCL ACCELERATED THE WEIGHTED MEAN FILTERING ALGORITHM

Figure 12 shows the performance of the optimized image weighted mean filtering algorithm on two different GPU computing platforms, AMD Radeon RX 5700 and NVIDIA GTX 1080, and its performance improvement relative to CPU serial computing. It is worth noting that the performance measurement of the OpenCL algorithm includes all the running time of the OpenCL system, including the initialization time of the OpenCL, the data transfer time between the CPU and GPU, and the running time of the kernel.

From Figure 12, it can be seen that the OpenCL weighted mean filtering parallel algorithms with different image sizes have achieved some performance improvement compared with the CPU serial algorithm on the AMD GPU platform and the NVIDIA GPU platform, respectively. Moreover, with the expansion of the computing scale, the acceleration ratio of the parallel algorithm is always more than 15 times. The good performance of GPU comes from the fact that it has many compute units and huge throughput to provide multipoint computing. It can make full use of the GPU's many computing resources to fully realize its performance. This is why the speedup has been maintained at a high level. Compared with the NVIDIA platform, the acceleration ratio of the AMD platform is smaller, the main reason is that the performance of the AMD GPU card used in this paper is not as good as the NVIDIA GPU card, and the performance of the parallel algorithm is affected to a certain extent.

VI. CONCLUSION

In this paper, the parallel acceleration of the image weighted mean filtering algorithm is completed by using the OpenCL heterogeneous computing platform. The part of image processing is parallelized by using the two-level parallel computing method of work-group and work-item in the two-dimensional index space. In the image convolution calculation, the two-dimensional convolution template and the corresponding image window data are used to do the convolution calculation. In order to speed up the convolution calculation, constant memory and synchronization instructions are used in the mean filter processing section to speed up the speed of data access. The experimental results show that the weighted mean filtering algorithm accelerated by OpenCL achieves the maximum speedup of 20.88 times and 20.29 times on the NVIDIA GPU and AMD GPU respectively, verifies the portability of the system performance, and solves the problem of hardware dependence of CUDA parallel acceleration. At the same time, compared with the weighted mean filtering parallel algorithm implemented in OpenMP multi-core CPU and CUDA, the maximum relative speedup of 18.52 times and 1.26 times is obtained respectively, which verifies the effectiveness of the algorithm. The acceleration results of this paper can be applied to image applications such as image measurement, image fusion, digital watermarking, edge detection, and so on.

There is room for performance improvement in the next step: The GPU used in this paper is NVIDIA GTX 1080, which is the fifth generation Pascal architecture. The GPU of the latest Ampere architecture has been interviewed, and the newer generation GPU can be selected for optimization to obtain a higher acceleration ratio. The single GPU working mode will be extended to a heterogeneous platform composed of multi-core CPU and multi-GPUs to meet the storage space and computing power requirements of a larger image test set.

REFERENCES

[1] U. Ozgunalp, "Robust lane-detection algorithm based on improved symmetrical local threshold for feature extraction and inverse perspective mapping," *IET Image Process.*, vol. 13, no. 6, pp. 975–982, May 2019.

[2] A. Dudhane and S. Murala, "Cardinal color fusion network for single image haze removal," *Mach. Vis. Appl.*, vol. 30, no. 2, pp. 231–242, Mar. 2019.

[3] L. Shang, S. Duan, L. Wang, and T. Huang, "SRMC: A multibit memristor crossbar for self-renewing image mask," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 12, pp. 2830–2841, Dec. 2018.

[4] A. K. Maurya, P. Agrawal, and S. Dixit, "Modified model and algorithm of LMS adaptive filter for noise cancellation," *Circuits, Syst., Signal Process.*, vol. 38, no. 5, pp. 2351–2368, May 2019.

[5] Y. Yuan, X. Yang, W. Wu, H. Li, Y. Liu, and K. Liu, "A fast single-image super-resolution method implemented with CUDA," *J. Real-Time Image Process.*, vol. 16, no. 1, pp. 81–97, Feb. 2019.

[6] D. Esposito, A. G. M. Strollo, E. Napoli, D. De Caro, and N. Petra, "Approximate multipliers based on new approximate compressors," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 12, pp. 4169–4182, Dec. 2018.

[7] J. Liu, B. Hu, and Y. Wang, "Optimum adaptive array stochastic resonance in noisy grayscale image restoration," *Phys. Lett. A*, vol. 383, no. 13, pp. 1457–1465, Apr. 2019.

[8] W. Witwit, Y. Zhao, K. Jenkins, and S. Addepalli, "Global motion based video super-resolution reconstruction using discrete wavelet transform," *Multimedia Tools Appl.*, vol. 77, no. 20, pp. 27641–27660, Oct. 2018.

[9] D. Liu and X. Chen, "Image denoising based on improved bidimensional empirical mode decomposition thresholding technology," *Multimedia Tools Appl.*, vol. 78, no. 6, pp. 7381–7417, Mar. 2019.

[10] J. Zabalza, J. Ren, J. Ren, Z. Liu, and S. Marshall, "Structured covariance principal component analysis for real-time onsite feature extraction and dimensionality reduction in hyperspectral imaging," *Appl. Opt.*, vol. 53, no. 20, pp. 4440–4449, 2014.

[11] T. Zeinab and A. Gholamreza, "Unsupervised texture-based SAR image segmentation using spectral regression and Gabor filter bank," *J. Indian Soc. Remote Sens.*, vol. 14, no. 4, pp. 1–10, 2015.

[12] D. Janecki, "Edge effect elimination in the recursive implementation of Gaussian filters," *Precis. Eng.*, vol. 36, no. 1, pp. 128–136, Jan. 2012.

[13] J. Fernandez-Berni, R. Carmona-Galan, and Á. Rodriguez-Vazquez, "Ultralow-power processing array for image enhancement and edge detection," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 59, no. 11, pp. 751–755, Nov. 2012.

[14] L. T. Bai, Y. W. Wu, J. C. Xie, P. C. Wen, and X. J. Liu, "Optimization methods for image haze removal based on multi-core DSP," *Electron. Opt. Control*, vol. 22, no. 10, pp. 14–18, 2015.

[15] D. B. K. Trieu and T. Maruyama, "Real-time color image segmentation based on mean shift algorithm using an FPGA," *J. Real-Time Image Process.*, vol. 10, no. 2, pp. 345–356, Jun. 2015.

[16] D.-H. Lee, J. Ahn, and W. Sung, "Parallel computation of adaptive filtering algorithms on multi-core systems," *J. Signal Process. Syst.*, vol. 69, no. 3, pp. 253–265, Dec. 2012.

[17] H. M. He, D. L. Qi, G. Y. Zhang, and J. L. Zhang, "Fast and efficient mean filtering algorithm for removing the salt and pepper noise," *Laser Infrared*, vol. 4, no. 4, pp. 469–472, 2014.

[18] D. Akgün, "A practical parallel implementation for TDLMS image filter on multi-core processor," *J. Real-Time Image Process.*, vol. 13, no. 2, pp. 249–260, Jun. 2017.

[19] M. Cheng, J. Z. Zhang, and Y. Yu, "Parallel image denoising algorithm based on multi-core MCMC," *Comput. Eng. Appl.*, vol. 50, no. 18, pp. 152–155, 2014.

[20] C. Salvatore, D. Pasquale, and P. Francesco, "3D data denoising via nonlocal means filter by using parallel GPU strategies," *Comput. Math. Methods Med.*, vol. 12, no. 3, pp. 1–15, 2014.

[21] T.-A. Nguyen, A. Nakib, and H.-N. Nguyen, "Medical image denoising via optimal implementation of non-local means on hybrid parallel architecture," *Comput. Methods Programs Biomed.*, vol. 129, pp. 29–39, Jun. 2016.

[22] G. L. Zou, C. J. Chen, and J. B. Hao, "Parallel implementation of wave sample data set for deep learning algorithm and its performance optimization," *Comput. Appl. Softw.*, vol. 34, no. 9, pp. 57–62, 2017.

[23] K. D. Subhra, M. Chandan, and B. Kumardeb, "GPU accelerated novel particle filtering method," *Computing*, vol. 57, no. 8, pp. 1–25, 2014.

[24] H.-H. Chang, Y.-J. Lin, and A. H. Zhuang, "An automatic parameter decision system of bilateral filtering with GPU-based acceleration for brain MR images," *J. Digit. Imag.*, vol. 32, no. 1, pp. 148–161, Feb. 2019.

[25] B. Chen, H. Chen, and X. H. Li, "Near real time linear stereo cost aggregation on GPU," *J. Image Graphics*, vol. 19, no. 10, pp. 1481–1489, 2014.

[26] X. W. He and X. Zhang, "A parallel algorithm of automatic time gain compensation for ultrasound imaging based on Fermi architecture," *Sci. Technol. Rev.*, vol. 30, no. 31, pp. 61–65, 2012.

[27] P. Lu, B. Sheng, S. M. Luo, X. Jia, and W. Wu, "Image-based non-photorealistic rendering for realtime virtual sculpting," *Multimedia Tools Appl.*, vol. 22, no. 8, pp. 1–18, 2014.

[28] H.-H. Chang and Y.-N. Chang, "CUDA-based acceleration and BPN-assisted automation of bilateral filtering for brain MR image restoration," *Med. Phys.*, vol. 44, no. 4, pp. 1420–1436, Apr. 2017.

[29] Q. Duan and H. Li, "High-speed parallel mean filter algorithm based on CUDA," *J. Xianyang Normal Univ.*, vol. 28, no. 4, pp. 52–55, 2013.

[30] H. L. Xia, Y. P. Chen, S. Y. Zhou, and Y. G. Tan, "Denoising method of steel sheet image based on CUDA," *Inf. Technol.*, vol. 40, no. 11, pp. 35–39, 2017.

[31] Y. N. Khalid, M. Aleem, U. Ahmed, M. A. Islam, and M. A. Iqbal, "Troodon: A machine-learning based load-balancing application scheduler for CPU–GPU system," *J. Parallel Distrib. Comput.*, vol. 132, pp. 79–94, Oct. 2019.

[32] K. Shata, M. K. Elteir, and A. A. El-Zoghabi, "Optimized implementation of OpenCL kernels on FPGAs," *J. Syst. Archit.*, vol. 97, pp. 491–505, Aug. 2019.

[33] Z. Wu, T. Alkhalifah, Z. Zhang, F. Alonaizi, and M. Almalki, "A new full waveform inversion method based on shifted correlation of the envelope and its implementation based on OPENCL," *Comput. Geosci.*, vol. 129, pp. 1–11, Aug. 2019.

[34] G. Tagliavini, G. Haugou, A. Marongiu, and L. Benini, "Optimizing memory bandwidth exploitation for OpenVX applications on embedded many-core accelerators," *J. Real-Time Image Process.*, vol. 15, no. 1, pp. 73–92, Jun. 2018.

[35] S. Nair, N. Somani, A. Grunau, E. Dean-Leon, and A. Knoll, "Image processing units on Ultra-low-cost embedded hardware: Algorithmic optimizations for real-time performance," *J. Signal Process. Syst.*, vol. 90, no. 6, pp. 913–929, Jun. 2018.

[36] A. S. Minkin, A. A. Knizhnik, and B. V. Potapkin, "GPU implementations of some many-body potentials for molecular dynamics simulations," *Adv. Eng. Softw.*, vol. 111, pp. 43–51, Sep. 2017.
[37] M. Korki and H. Zayyani, "Weighted diffusion continuous mixed p-norm algorithm for distributed estimation in non-uniform noise environment," *Signal Process.*, vol. 164, pp. 225–233, Nov. 2019.
[38] I. Erer and N. H. Kaplan, "Fast local SAR image despeckling by edge-avoiding wavelets," *Signal, Image Video Process.*, vol. 13, no. 6, pp. 1071–1078, Sep. 2019.
[39] F. Duan, Y. Pan, F. Chapeau-Blondeau, and D. Abbott, "Noise benefits in combined nonlinear Bayesian estimators," *IEEE Trans. Signal Process.*, vol. 67, no. 17, pp. 4611–4623, Sep. 2019.
[40] C. Liu, Z. Zhang, and X. Tang, "Sign normalised Hammerstein spline adaptive filtering algorithm in an impulsive noise environment," *Neural Process. Lett.*, vol. 50, no. 1, pp. 477–496, Aug. 2019.
[41] V. N. Karnaukhov and M. G. Mozerov, "Fast non-local mean filter algorithm based on recursive calculation of similarity weights," *J. Commun. Technol. Electron.*, vol. 63, no. 12, pp. 1475–1477, Dec. 2018.
[42] F. Huang, J. Zhang, and S. Zhang, "Affine projection versoria algorithm for robust adaptive echo cancellation in hands-free voice communications," *IEEE Trans. Veh. Technol.*, vol. 67, no. 12, pp. 11924–11935, Dec. 2018.
[43] W. Wang, J. Sun, R. Dong, Y. Zheng, and Q. Hua, "The development of a high accuracy algorithm based on small sample size for fingerprint location in indoor parking lot," *IEICE Trans. Commun.*, vol. E101.B, no. 12, pp. 2479–2486, Dec. 2018.
[44] Z. Zhu, X. Zhou, L. Deng, K. Wang, and B. Zhou, "Quantitative analysis of geophysical sources of common mode component in CMONOC GPS coordinate time series," *Adv. Space Res.*, vol. 60, no. 12, pp. 2896–2909, Dec. 2017.
[45] S. Jayaprakasam, X. Ma, J. W. Choi, and S. Kim, "Robust beam-tracking for mmWave mobile communications," *IEEE Commun. Lett.*, vol. 21, no. 12, pp. 2654–2657, Dec. 2017.

HAN XIAO was born in Wuhan, Hubei, China, in 1970. He 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 the School of Information Engineering, Zhengzhou University. Since 2012, he has been a level 3 Professor with the School of Information Science and Technology, Zhengzhou Normal University. 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.

BAOYUN GUO was born in Chuzhou, Anhui, China, in 1986. She received the Ph.D. degree in photogrammetry and remote sensing from the School of Remote Sensing and Information Engineering, Wuhan University, China, in 2013. She is currently a Lecturer with the School of Civil and Architectural Engineering, Shandong University of Technology. Her main research interests include industrial photogrammetry and computer vision.

HONGYAN ZHANG was born in Zhengzhou, Henan, China, in 1982. She received the master's degree in computer application technology from the School of Information Engineering, Zhengzhou University, China, in 2008. She is currently a Lecturer with the School of Information Science and Technology, Zhengzhou Normal University. Her main research interests include image processing and high performance computing.

CAILIN LI was born in Anqing, Anhui, China, in 1985. He 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 with the School of Civil and Architectural Engineering, Shandong University of Technology. His main research interests include digital photogrammetry, computer vision, and digital image processing.