Research for High Speed Image Processing Programming Method on Combined Environment of CUDA and OpenCV

Satomi Kameyama, Yasuyuki Miura
Shonan Institute of Technology

Abstract-- **When the OpenCV GPU method and the existing CUDA kernel function are mixed, memory copying occurs frequently between the main memory and the GPU device memory. In order to solve this problem, we tried to reduce the effort of programmer and reduce memory copies by using modules for implementing GPU in OpenCV environment. As the experimental result of the proposed method, it turned out that the predetermined processing can be handled without a serious influence to the whole system.**

I. INTRODUCTION

Security camera is a crime prevention device that records the movement of people and things, and has a crime prevention effect that leaves video evidence of crime. Improving the performance of moving-object detection makes it possible to incorporate more complex and accurate processing. However, since the image processing capability of the current apparatus is limited, it is necessary to develop a higher performance apparatus. When developing a moving object detection program using the GPU, problems arise during processing if a GPU method in the library of OpenCV and a CUDA kernel function are mixed. Specifically, when a memory access occurs between the host memory and the device memory, a memory copy is executed. If this copy process is executed frequently, it seems to affect the processing of the whole program.

In this research, we revise the data structure of the moving object detection software using the CUDA implementation module of the OpenCV library and compare the processing speed of the program. In addition, we will consider reducing the effort of development.

II. MOVING-OBJECT-DETECTION

As a process of the moving-object detection, image frames from the video camera are captured, grayscale of the captured image is applied, and output to the screen. Thereafter, the image data is copied from the PC memory to the GPU memory, and noise removal processing is executed. After the processing, block matching processing is executed, the image is copied to another memory of the GPU, and the image from the GPU memory is copied to the CPU memory. Then, the processing result is output and it is repeated from the acquisition of the frame. The operation sequence of moving-object detection is as follows.

1. Frame capture
2. Grayscale
3. Screen display
4. Copy data from PC memory to GPU memory
5. Denoising
6. Block matching
7. Copy image data from GPU memory to main memory

III. OPENCV CORE MODULE

A. *cv::Mat class*

The cv::Mat class handles image data under the OpenCV environment and can be used to store real or complex vectors and matrices, grayscale or color images, histograms, and so on. The generation of the data structure using the cv::Mat class is described as shown in Fig 1. Fig 1 shows the code that creates an instance for storing image data called Amat. Size represents the size of the image data. *Type* specifies the type of the element. For example, to make it "unsigned char" type, CV_8U is written in there. *Scalar* represents a vector of 4 elements. It is mainly used to represent RGB color.

```
cv::Mat Amat( Size, Type, Scalar );
```

Fig 1. Create instances of Mat class.

B. *GpuMat class*

GpuMat is a basic structure class of GPU memory. Like the cv::Mat class, it supports the provision of image/matrix data structure, array operation, etc. It is a class supported to realize CUDA implementation in OpenCV environment. There are two limitations. It supports only two dimensions, and there is no function that returns a reference to the data. To generate a data structure using the GpuMat class, Fig 2 is specified. An example of creating an instance having the same value as the data on the cv::Mat side is shown in Fig 3.

```
cv::cuda::GpuMat d_Amat(Size, Type, Scalar)
```

Fig 2. Create instances of GpuMat class.

```
cv::cuda::GpuMat d_Amat(Amat);
```

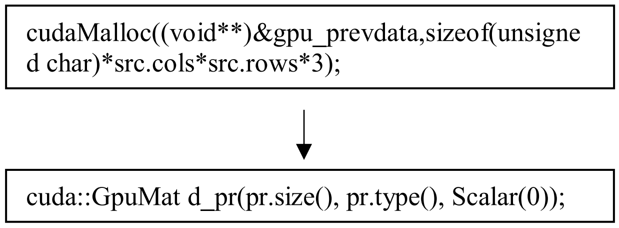
Fig 3. Creating an instance with Amat data.

IV. Proposed Method

A. Revision of basic structure

Programs mixed with OpenCV GPU method and existing CUDA kernel functions have complicated data structures. Therefore, it is necessary to unify the data structure to the basic structure of OpenCV. It is revised to data structure called GpuMat prepared for CUDA implementation of OpenCV module to execute CUDA kernel function. In the official document of OpenCV, it is stated that GpuMat class can be converted to cuda::PtrStepSz and cuda::PtrStep. It is also stated that it can be passed directly to the kernel. As you can see, it is possible to pass instances of GpuMat class to the CUDA kernel by using GpuMat.

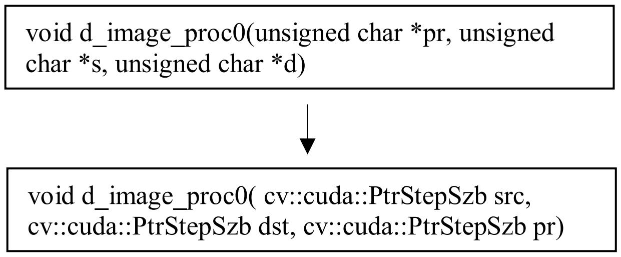
Fig 4 shows a code that changes the data structure used for kernel functions and image data in existing programs. And Fig 5 shows a diagram of changing from the image data received on the CUDA kernel to cuda::PtrStep.



```
cudaMalloc((void**)&gpu_prevdata,sizeof(unsigne
d char)*src.cols*src.rows*3);
```

```
cuda::GpuMat d_pr(pr.size(), pr.type(), Scalar(0));
```

Fig 4. Diagram of data structure change.



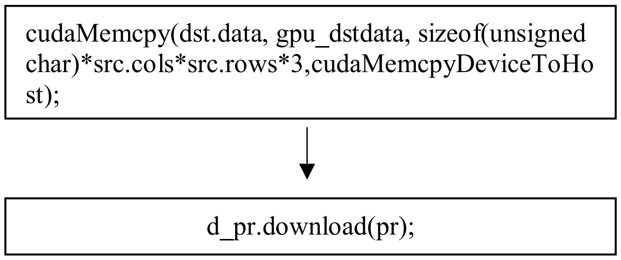
```
void d_image_proc0(unsigned char *pr, unsigned
char *s, unsigned char *d)
```

```
void d_image_proc0( cv::cuda::PtrStepSzb src,
cv::cuda::PtrStepSzb dst, cv::cuda::PtrStepSzb pr)
```

Fig 5. Changing the received arguments.

B. Revision of memory copying method

Previously processed image data was copied from device memory to host memory using CUDA standard copy function, but revised to copy function for OpenCV. As a result, image data is copied in the cv::Mat class. The above method is shown in Fig 6.



```
cudaMemcpy(dst.data, gpu_dstdata, sizeof(unsigned
char)*src.cols*src.rows*3,cudaMemcpyDeviceToHo
st);
```

```
d_pr.download(pr);
```

Fig 6. Image data copy.

V. PERFORMANCE EVALUATION

As a performance evaluation, the results of the average processing time per 1 frame of the program before and after changing are plotted to table. The frame capture time depends on the processing capacity of the camera, so it is irrelevant in this research. The CPU in our environment was core i7-2600k 3.40GHz (8 cores), and the main memory was 8192MB. The GPU was NVIDIA Geforce GTX TITAN Black with 4095MB memory.

Table 1. The result of processing time.

Process	Before	After
Frame capture	37.812ms	37.202ms
Grayscale	0.170ms	0.175ms
Screen display	0.404ms	0.385ms
CPUtoGPU	0.201ms	0.785ms
Denoising	0.644ms	0.474ms
Block matching	5.006ms	5.318ms
GPUtoCPU	0.580ms	0.591ms
Screen display	0.961ms	0.97ms
Average per frame	7.966ms	8.698ms

VI. Experimental Result

The result is shown in Table 1. The average processing time of real-time processing in one frame before change was 7.966 ms, and after change was 8.698 ms. The result of changing the process of copying from CPU memory to GPU memory was 0.584 ms slower than before change. The reason of this result seems to be that an instance of the cv::Mat class is newly created in the instance of the class of GpuMat.

VII. Conclusion

As a result of measuring the processing speed by programming with a combination of OpenCV and CUDA, it was found that it operates without serious affecting the entire system during the processing of the program, as compared with before the change. It is thought that reduction of effort of system development can be realized in the future by using the CUDA module function for filtering processing etc, which was made in the existing program inside. And, since the handling of the image data is unified to the OpenCV side, it is expected that the specification change of the program becomes easy. As a future work, we are considering the development of image processing by combining face recognition and *Camshift* etc.

Reference

[1] Yasuyuki Miura, Yuta Fujii, 'The Examination of the Image Correction of the Moving-Object Detection for Low Illumination Video Image, IEEE International Conference on Consumer Electronics – Taiwan (IEEE 2015 ICCE-TW), pp.33-34, 2015.06.

[2] Yasuyuki Miura et.al., The Development of the GPU based Experiment System for the High-Speed Moving-Object Detection for Low Illumination Video Image, IPSJ SIG Technical Report, Vol.2015-HPC151, No.11, pp.1-7, 2015.09.