Computing, Performance and Communication Systems (2020) Vol. 4: 9-13 **DOI: 10.23977/cpcs.2020.41002**
Clausius Scientific Press, Canada **ISSN 2371-8889**

Study on the Collaborative Acceleration Method Between CPU and GPU in Image Processing of Mask Detectio

Hongbin Wei[1, 2], Song Hu[1, 2, a, *]

[1]*Institute of Optics and Electronics Chinese Academy of Sciences, Chengdu, China*

[2]*University of Chinese Academy of Sciences, Beijing, China*

[a]husong@ioe.ac.cn

*Corresponding author: husong@ioe.ac.cn

Keywords: CUDA, CPU, Mask image processing, GPU

Abstract: The image processing methods of the current big data volume are based on parallel computing to improve the processing speed. There are two ways of mainstream parallel image processing, one is to use DSP or FPGA parallel processing, and the other is to use GPU-based CUDA parallel distributed system. DSP or FPGA parallel image processing mode can realize the complex operation, the fast and low power consumption, but the development is difficult, the developer needs to be familiar with the hardware and software knowledge, write algorithms for different hardware structure, program portability is very poor and the development cycle is long. In GPU-based CUDA parallel processing system, the GPU is responsible for performing highly threaded image parallel processing tasks, the CPU is responsible for logical image processing and serial computing, and the CPU and GPU work together. GPU as a coprocessor, low power consumption, large memory and transmission capacity, currently fully support C and C language, easy to develop and because of the hardware structure fixed algorithm portability is high. The GPU parallel processing technology, with its unique multi-threaded architecture acceleration model, plays an important role in improving the speed of mask defect detection and processing.

1. Introduction

Mask detection compared to ordinary print defect machine visual detection, there are three problems to be solved: the accuracy of detection is higher, usually in several microns and nanometerlevels, and the defect detection accuracy of the printing can reach the millimeter level; To make the defect detection efficiency is reduced accordingly, improving the detection speed at the same time to ensure accuracy is an inevitable difficulty [1].

The image processing methods of the current big data volume are based on parallel computing to improve the processing speed. There are two ways of mainstream parallel image processing, one is to use DSP or FPGA parallel processing, and the other is to use GPU-based CUDA parallel distributed system. DSP or FPGA parallel image processing mode can realize the complex operation, the fast and low power consumption, but the development is difficult, the developer needs to be familiar with the hardware and software knowledge, write algorithms for different hardware structure, program portability is very poor and the development cycle is long [2]. In GPU-based CUDA parallel processing system, the GPU is responsible for performing highly threaded image parallel processing tasks, the CPU is responsible for logical image processing and serial computing, and the CPU and GPU work together. GPU as a coprocessor, low power consumption, large memory and transmission capacity, currently fully support C and C language, easy to develop and because of the hardware structure fixed algorithm portability is high [3]. The GPU parallel processing technology, with its unique multi-threaded architecture acceleration model, plays an important role in improving the speed of mask defect detection and processing.

2. CPU-GPU Task Assignment Design

In the mask detection image processing system, the relatively simple and dense calculation algorithms such as image filtering enhancement, second-valueization, registration and morphological processing are theoretically applicable to improved parallel implementation in GPU, while CPU manages parallel processing tasks, makes logical judgment of processing results, and finally realizes the detection of mask graphic defects.

The platform used in this article is a joint compilation environment for OpenCV and CUDA, the CPU-GPU image processing process is: the image data is crawled to buff, converted to Mat format stored in CPU memory, the writer and GPU kernel functions, when using CPU processing, direct processing of Mat data, when the CUDA GPU processing, the data into Mat GPU format, transferred to GPU memory for multi-threadprocessing, At the end of this year, the GPUMat format data is transferred to Mat data transfer to memory. The GPU module in OpenCV is easy to use, multi-class function acceleration can be achieved, combined with CUDA according to GPU hardware architecture and data type, the image can be processed in the GPU data stream and then task allocation, in the large amount of data and need to repeat the step of the acceleration ratio is high [4].

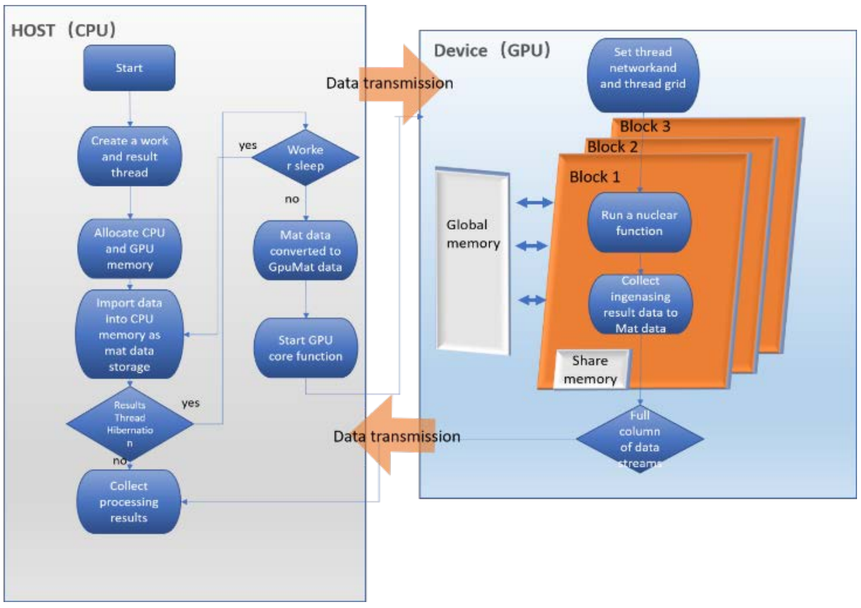
HOST（CPU）
Start
Create a work and result thread
Allocate CPU and GPU memory
Import data into CPU memory as mat data storage
Results Thread Hibernation
yes
no
Collect processing results
yes
Worker sleep
no
Mat data converted to GpuMat data
Start GPU core function
Data transmission
Data transmission
Device（GPU）
Set thread networkand and thread grid
Block 3
Block 2
Block 1
Global memory
Run a nuclear function
Collect ingenasing result data to Mat data
Share memory
Full column of data streams

Figure 1. Image processing flow and task allocation using CPU+GPU

The CPU in the CUDA model is mainly responsible for image data loading and initialization, including image data format conversion [5], memory allocation, data transfer and memory release.

In experiments, data transmission is often the most time-consuming process, and there are two ways to solve this problem to optimize processing efficiency: first, to minimize the number of data transfers in GPU and CPU at design time, to make more use of the faster cache and register memory in GPU memory, and second, to use multi-task parallel hidden data transmission access time. The tasks assigned to the CPU and GPU are shown in Figure 1.

3. CPU Parallel Task Design

We combine the flow idea of CUDA parallel architecture with the queue idea to include the team thinking, build the queue for image data transfer, and introduce CPU multithreaded hidden data transfer and access time. The multithreaded design of CPU in the mask defect detection system is divided into three threads of detection, tracking and result, which are responsible for the three tasks of loading image data from disk to CPU memory, CPU memory data into GPU tracking processing progress and receiving GPU outgoing data, and is responsible for the task of calculating data transfer data and processing results for GPU [6].

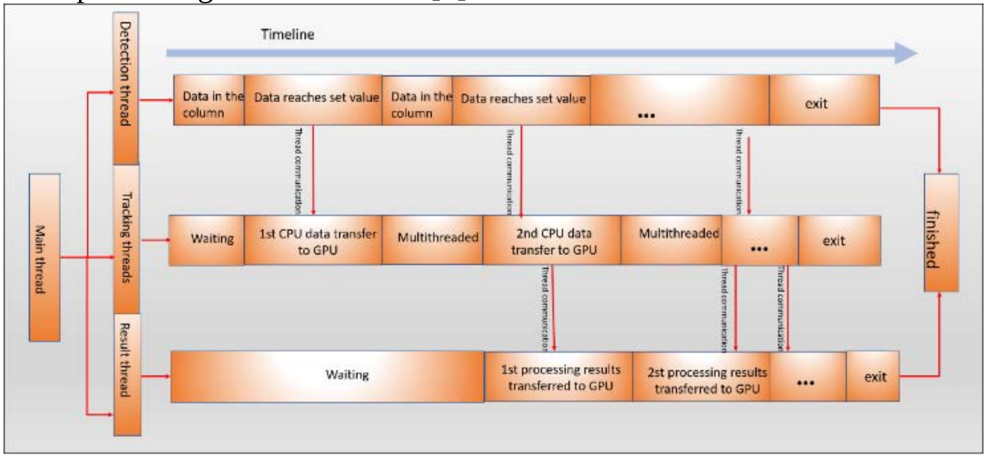

Figure 2. CPU parallel task timeline

As shown in the figure timeline, the CPU parallel task is handled by: detection thread loads mask image data into CPU memory, loads the current queue in the queue order, the picture data starts to load into the next queue; When GPU processing completes the number of images reaches the set threshold the data results are returned to CPU memory. In the above steps, the detection, tracking, and result threads have interthreaded communication and set mutual exclusions to ensure synchronization of processing. The CPU used in this system is a quad-core i7 processor, each single-core can realize single-threaded tasks, realize the maximum utilization of CPU resources, greatly improve the processing speed of mask defect detection.

4. GPU thread inger design

CUDA thread grids are designed to prepare for the decomposition of data and tasks into parallel threads and thread blocks. Grids, thread blocks, and threads form the entire parallel computing architecture in CUDA, multiple thread blocks form a grid, multiple threads form thread blocks, and threads in the same thread block can communicate using shared memory. Thread grids can be customized according to data or algorithms, the definition format can be 1D 2D and 3D indexes [7], different data or tasks are assigned to separate SM, so that memory data and programs are mapped. The difference in the way thread grid is divided directly will directly affect the program acceleration effect, so thread grid design is the key point of CUDA program design. The following rules are often followed in thread grid designs:

(1) Global memory access merge: To reduce latency, try to merge memory requests from connected threads.

(2) The smallest number of global memory reads and writes: global memory access speed is much slower than processing speed, should be as little access to global memory as possible, as much as possible to make use of shared memory and registers.

(3) Hide memory access time, make full use of thread resources: try not to use small thread blocks, the number of thread blocks designed as far as possible for the number of SM integer times, because the cell thread bundle is a collection of 32 threads, the number of threads designed within the thread block should be 32 integer times, the program try to avoid using logic to judge the structure and conditional branch structure.

Table 1 GPU device thread information table

Serial number	Name	Number
1	TotalGlobalMem	4234967295
2	sharedMemPerBlock	49152
3	RegsPerBlock	65536
4	warpSize	32
5	maxThreadsPerBlock	1024
6	maxThreadsDim	(1024,1024,64)
7	maxGridSize	(2147483647,65535,65535)

GPU device information is shown in Table 1: there are a maximum of 65535 thread blocks per grid, a maximum of 1024 threads per thread block, and 10 SMs.

According to the design rules and image size we can segment the image into small images by various schemes, so as to carry out the parallel processing of data chunks. For example, when working with a single mask image with a resolution of 2048 by 1536, the dimension of the thread block is first set as blockdim (32,32), i.e. 32 and 32 threads in the x direction and 32 threads in the direction of y, for a total of 768 threads;

5. Conclusion

In the previous section, we discussed how to design thread slots for data to make full use of GPU resources for parallel processing, and then we delved into the strategy of further optimizing the program in the CUDA parallel processing model [8].

There is a two-tier parallel pattern of coarse and fine-grained in CUDA. In the GPU model used in this article, each thread block can execute up to 1024 threads, with thread bundles as the smallest execution unit for thread bundles, and each thread bundle consisting of 32 threads. The instruction address of the thread bundle in the same thread block is the same, and can be communicated through shared memory while executing in parallel. The parallelism between different thread blocks we call coarse-grained parallelism, the parallelness of different thread beams of the same thread block that can communicate We call fine-grained parallelism, which constitute the two-layer task parallel pattern of CUDA coarse-grained.

Before parallel tasks are performed in CUDA, the task allocation unit assigns thread grids to the GPU schema, and the thread blocks correspond to the SM units. Factors that affect whether a thread block can be allocated to an SM are the amount of shared memory that thread blocks need, the number of registers used for each block, and so on, so we can control the number of threads in the block and the size of the required memory, so that the amount of tasks in each SM is as consistent as possible, and the resource utilization is fully improved. SM's computing power and memory access speed are limited by hardware conditions are limited, in SM at the same time there will generally be multiple blocks of thread bundles running, if a single SM thread block too many reality is that part of the resources can only wait, a great impact on processing speed.

We introduce a disorderly execution model when SM executes programs, which can solve such problems and improve processing efficiency: while some blocks of thread bundles are accessing operations such as reading and writing memory, while others, thread bundles of threads of threads use SM computing resources to perform simple disorderly operations between different blocks of SM. Extending this idea to the level of task processing, we can try to perform in a disorder of different task kernel functions, when one task's kernel function is in reading and writing memory, and the kernel function of another task uses computing resources for operations, which can greatly improve GPU utilization.

Combined with CPU's multithreaded parallel planning, we can greatly improve mask image processing speed by using CPU task parallel, CUDA coarse-grained architecture model and disorderly execution mode. CPU multithreaded transfer of image data to GPU memory in batches, start GPU core function and receive result data, data access and processing of different thread blocks in CUDA is staggered asynchronous execution, making the overall data access and processing efficiency the highest, will take the longest proportion of CUDA internal data access time hidden. In most of the program processing time, the data from the CPU to the CPU, GPU core function start, GPU internal data access and GPU internal processing calculation are synchronized, improving the speed of mask image processing algorithm.

References

[1] Luo Jingjing, Han Baoan. Image matching based multi ship image mosaic method [J]. Ship science and technology, 2019, 41 (16): 61-63.

[2] Cheng Xixi, Zhang Yanling, Tian Junwei. A new fast corner detection method based on template matching [J / OL]. Computer Engineering: 1-6 [2019-10-22]. Http://kns.cnki.net/kcms/detail/31.1289. tp.20190719.1724.010.html.

[3] Zhang Jinrong, Chen xunlin, Luo Yanqi, Zhang Panfeng. Fast template matching of integrated circuit online detection based on bicubic interpolation algorithm [J]. Science and technology and engineering, 2019, 19 (19): 185-189.

[4] Sheng Mingwei, Tang Songqi, Wan Lei, Qin Hongde. Overview of 2D image mosaic technology [J]. Navigation and control, 2019, 18 (01): 27-34 + 96.

[5] Yang Fan. Research on real-time image acquisition and splicing technology of industrial online visual inspection system [D]. China University of science and technology, 2018.

[6] Gong miaolian. Research on image registration and splicing technology based on feature points [D]. Beijing University of Posts and telecommunications, 2018.

[7] Jia Di, Yang Ninghua, sun Jingguang. Template selection and matching of image pair matching [J]. Chinese Journal of image graphics, 2017, 22 (11): 1512-1520.

[8] Wang Juan, Shi Jun, Wu Xianxiang. Overview of image mosaic technology [J]. Computer application research, 2008 (07): 1940-1943 + 1947.