JPEG2000 image compression method based on GPGPU

Wang Weiling
School of Science, Changchun Institute of Technology
Changchun, 130012, China
wang_weiling_yx@126.com

***Abstract*—In order to improve the compression speed of JPEG2000, the JPEG2000 compression standard is analysised and it concluded that the part data of the core algorithm that is DWT in JPEG2000 are independent from each other, so it is very suitable for parallel processing. CUDA (Compute Unified Device Architecture) is a latest software and hardware exploitation platform released by NVIDIA which is very suitable for large-scale data parallel computing.Using CUDA technology on general purpose graphic process unit(GPGPU) could speed up DWT algorithm parallelly and the program is optimized based on the characteristics of GPGPU storage space. The obtained experimental results show the DWT algorithm that is optimized by CUDA parallelly can improve the computing speed.**

Keywords-JPEG2000; DWT; GPGPU; parallel computing; CUDA

I. INTRODUCTION

With the wide spread of multimedia technology in the field of computer science, image compression technology has become one of the key technologies in the field of modern digital image transmission, processing and storage. Image compression technology has been played an important role in both mobile communication and network transmission.JPEG2000 is a new still image compression standard which proposed on the basis of JPEG. Compared with the JPEG compression standard, JPEG2000 is optimized not only in the compression performance, so that the image data can be compressed with higher compression ratio. But also in the advantage of supporting both lossy and lossless compression. As Image pixels can be seen as two-dimensional array, and calculation of the two-dimensional array is equivalent to compute a large number of irrelevant data[1], especially for the original bitmap with high quality.It will take the traditional CPU a lot of time to process because of its serial architecture. This can not fulfill the real-time requirements of the image compression in modern multimedia technology. However, with the release of General Purpose graphics processors unit (GPGPU), except the graphics processing architecture of the traditional GPU, GPGPU has also increased the parallel computing architecture which make it possible for computing-intensive processing and high-strength parallel accelerated computing. NVIDIA Corporation provided a new hardware and software development platform named CUDA (Compute Unified Device Architecture)for its GPGPU. Compared with the implementation on CPU, the optimization of the core algorithm in JPEG2000 on GPGPU by CUDA technology has been significantly improved in both speed and efficiency of image compression.

II. JPEG2000 IMAGE COMPRESSION METHOD

Compared with JPEG, JPEG2000 image compression standard has a further improvement in the algorithm. First of all, in order to reduce the redundant information among the image data, JPEG2000 chooses discrete wavelet transform (DWT) instead of discrete cosine transform (DCT) in JPEG.This could avoid the box noise which has brought by JPEG in the low bit rate case. Secondly, about the entropy coding algorithm, JPEG2000 selected embedded block coding with optimized truncation (EBCOT) algorithm to replace the Huffman encoding algorithm in JPEG.

JPEG2000 core coding system mainly consists of six modules which is shown in Fig.1. Firstly, in order to get the wavelet coefficients, the result of preprocess for original image data need to go through the wavelet transform module.Then, according to the specific requirements to quantify the transformed wavelet coefficients. Next, these quantized wavelet coefficients will be divided into code blocks to be embedded coding independently and form the codestream. These codestreams are organized hierarchically according to the rate-distortion optimization principle. Finally, these different quality layer were packaged

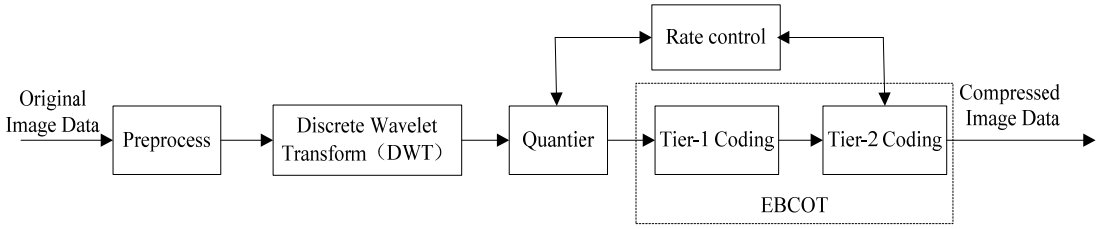

Fig.1 JPEG2000 core encoder block diagram

Published by Atlantis Press, Paris, France.

according to the specific codestream format to output. The description above can be called the compression process of entire image .

III. GPGPU AND CUDA

With the advent of the 3D era, the computing power of traditional CPU can not fulfill the requirement of the 3D graphic data computation. To this end, GPU has emerged. GPU is designed for graphic data computation specially. Compared with CPU, GPU has lots of advantages such as the high floating-point performance, high bandwidth and the highly efficient parallel computing. But such a powerful computing capability is used only for graphics rendering which is undoubtedly a waste of computing resources. In order to make full use of the GPU computing capability, not only for graphics rendering, but also for fulfilling the real-time requirement of other scientific computing fields, general purpose GPU (GPGPU) came into being, and has achieved great results.The hardware of GPGPU use a single instruction multiple data (SIMD) architecture. And it was a perfect combination of graphics processing architecture and parallel computing architecture[2]. This kind of architecture makes GPGPU can not only do their own work as a graphics card for graphics rendering, but also make use of other non-graphical areas extensively for exerting its powerful computing capacity fully. A hardware module of GPGPU consists of multiple stream multiprocessors (SM) which is shown in Fig.2. Each stream multiprocessor contains 8 stream processors (SP), two special function units (SFU) and

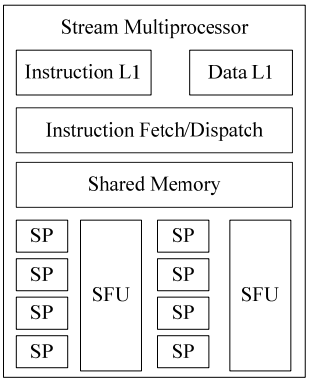

Fig.2 the GPGPU hardware SM structure

some memory resources such as shared memories and registers. According to the features and access speed of different memories, allocating the size of memory space rationally is the key to improve the performance of GPGPU parallel accelerated computing.

CUDA is a heterogeneous development platforms which is designed specifically by NVIDIA Corporation for its GPGPU production in June 2007. Since then, the fate of GPGPU parallel accelerated computing has completely changed. CUDA provides interfaces to the hardware access directly. Different from the development of traditional GPGPU, modern GPGPU development does not require the help of the graphics API like Open GL and Direct X. At the same time, the widely used C language was expanded by CUDA. And the difficulties of programming were further reduced by CUDA technology. So the developers can easily transit from the C language application development to a GPGPU application development.CUDA provides the single instruction multiple threads (SIMT) heterogeneous programming execution model which corresponds to the SIMD architecture of GPGPU[3]. This model is shown in Fig.3.

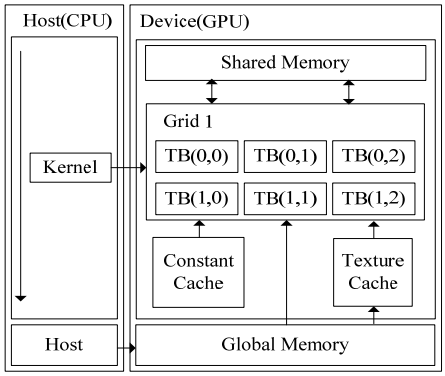

Fig.3 CUDA heterogeneous programming model

The hierarchical thread structure of CUDA includes threads, thread blocks and grid. Each grid consists of a certain number of thread blocks, and each thread blocks contains up to 512 threads. A CUDA program is composed of the program running on Host (CPU) and the program running on Device (GPU). The host-side executes the serial commands to allocate the task, and the device-side as co-processor executes parallel accelerated computing section. The program which was executed on device was called the kernel function. Each kernel function executes in a Grid. A simple device program was required to complete the following two processes. Firstly, the data to be processed need to be copied from the host memory to the global memory of device before calling the kernel function. Secondly, when the calculations are completed, the results need to be returned from the global memory to the host memory.

In order to improve the computing speed of the entire task, the substance of CUDA parallel accelerated computing is to divide a task into multiple independent partial tasks[4]. These partial tasks were processed by thousands of threads simultaneously. Therefore, the image data is divided into some independent data blocks and then processed by CUDA technology can improve the speed of the JPEG2000 image compression.

A. DWT on the GPGPU

There are two discrete wavelet transform algorithms in JPEG2000 standard: the 5/3 integer wavelet lifting algorithm and the 9/7 floating-point wavelet lifting algorithm. The 5/3 wavelet lifting algorithm is fit to both lossy and lossless compression and it is described as follows:

$$y(2n+1) = x(2n+1) - \lfloor (x(2n) + x(2n+2))/2 \rfloor$$
$$y(2n) = x(2n) + \lfloor (y(2n-1) + y(2n+1) + 2)/4 \rfloor \quad (1)$$

In the case of low bit rate, the 9/7 floating-point wavelet lifting algorithm can play the most superior performance and it is recommend in lossy compression. Compared with the5/3 lifting wavelet algorithm, the 9/7 wavelet lifting algorithm is more complex:

$$\begin{aligned} y(2n+1) &= x(2n+1)+\alpha[x(2n)+x(2n+2)] \\ y(2n) &= x(2n)+\beta[y(2n-1)+y(2n+1)] \\ y(2n+1) &= y(2n+1)+\gamma[y(2n)+y(2n+2)] \\ y(2n) &= y(2n)+\delta[y(2n-1)+y(2n+1)] \\ y(2n+1) &= -K\times y(2n+1) \\ y(2n) &= (1/K)\times y(2n) \end{aligned} \quad (2)$$

Where,

$$\alpha = 1.486134342,\ \beta = -0.042980118,\ \gamma = 0.882911075 \quad (3)$$
$$\delta = 0.443506852,\ K = 1.230174105$$

B. Core algorithm on CUDA

The wavelet transform algorithm in JPEG2000 shows that the one-dimensional transform need a large amount of manipulations about data[5]. So the basic row transform and column transform need to be designed into kernel function which is called through the device to complete. And other parts of the task are handed by host to complete. From the analysis above, it can come up with a block diagram which is shown in Fig.4:

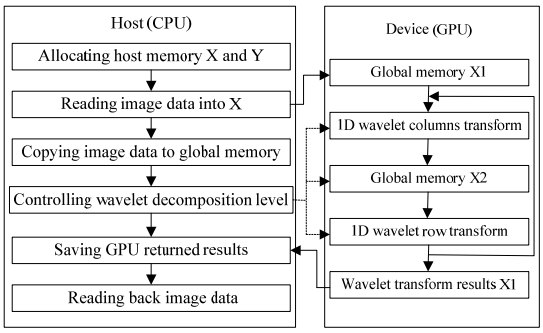

Fig.4 DWT task allocation block diagram

The specific steps to implement the process mentioned above are as follows:

1. Allocated host memory space X and Y on host to store input image data and output compressed image data respectively.After that copying the image data to the host memory. Then two identical global memory spaces X1 and X2 was allocated on device by calling the library function cudaMalloc().
2. For the subsequent wavelet column transform calculation, the image data need to be copied from host memory to global memory space X1 that has already set up with a suitable size on device by calling the library function cudaMemcpy ().
3. Initialized the input image data, parameters and shared memory size.Then copied the segmented image data from global memory X1 to the shared memory that has already set up.After that the image data was operated by lifting wavelet column transform. Take the 9/7 wavelet transform for example, the flow diagram is showed in Fig.5.

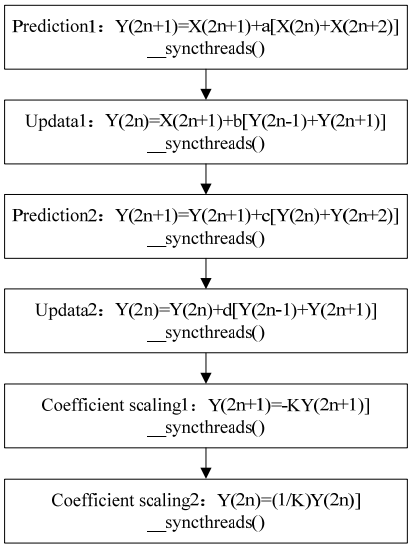

Fig.5 9/7 wavelet lifting kernel processes

4. The synchronous function must be used at the end of each step to ensure the correctness of the results. And then the results need to be stored sequentially in the global memory space X2 which has already set up.
5. The results of the previous step was operated according to the same method ——1D wavelet row transform and then stored the results in the global memory space X1 which has already been vacant .
6. The processing procedure of wavelet lifting transform need to be repeated depended on the wavelet decomposition level until the entire procedure has completed. Finally, the results of image data which were outputted need to be returned back to the host memory of the host from the global memory of the device. Then release the device memory space and the host memory space.

IV. OPTIMIZATION OF ACCESSING THE SHARED MEMORY

As the speed of accessing the global memory is very slow. If the image data in the global memory was accessed repeatedly, it will cut down the efficiency of the program to execute significantly. However, the speed of accessing the shared memory is as fast as the speed of accessing the register[6]. So the optimization strategy about accessing the shared memory was selected to accelerate the DWT algorithm. But when multiple threads access the data in the same location of the shared memory, it will bring bank conflict which can affect the performance of the program to execute. So the bank conflict must be avoided during the optimizing procedure.

Firstly, the image data in the global memory is divided into (n +1) × (n +1) data blocks named DB. And at the same time the same number of the thread blocks were defined and named TB for short. These thread blocks were required to correspond to these data blocks one-to-one.

The corresponding relationship is shown in Fig.6. The size of the shared memory space was defined for each thread block to store the image data for each thread to calculate.

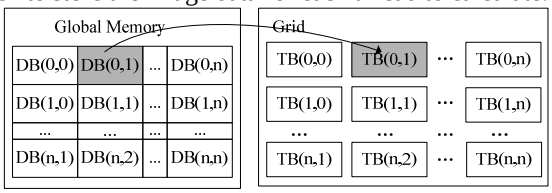

Fig.6 shared memory data access optimization

The advantage of the shared memory is its fast access speed. According to the description above, copy the original image data which has been divided into blocks from the global memory to the shared memory. Then each thread in per thread block calculate the corresponding data in the shared memory directly. To do like the description above will avoid the bank conflict that may cause. In addition, when optimizing the CUDA programs, make sure that it is required to comply with the coalesced memory access condition to access global memory. Coalesced memory access conditions is also one of the most important factors in optimizing CUDA program. Apart from this, optimizing CUDA program also need to consider calculation accuracy, latency, the amount of calculation about data and other factors at the same time.

TABLE I. COMPARISON OF THE TEST TIME ABOUT DWT ON CPU AND GPGPU

Pixels	640*480	1280*860	2592*1944	3460*2318	5680*4504
CPU(ms)	47	328	904	2198	4977
GPGPU(ms)	13.896	34.671	67.708	148.61	327.53
Speedup	3.38	9.46	13.35	14.79	15.20

V. TEST RESULTS

The hardware and software test environment requirements are as follows:

- The CPU hardware environment:

Intel E7400 Core 2 Duo dual-core 2.80GHz CPU, clock 2800MHz, host memory 2GB;

- The GPU hardware environment:

NVIDIA GeForce GTX 560 Ti Graphic Card, CUDA core 384, Graphics clock 822 Mhz, Processor clock 1645 MHz, memory clock （MHz）4008 Gbps, standard memory config 1024 MB, memory interface width256-bit, memory bandwidth 128GB/sec,computing capability 2.1, bus support PCI-E 2.0x16;

- Programming environment:

GPU hardware drivers Version 301.42, CUDA Version 4.1, Windows 7 OS, Visual Studio 2010.

The JPEG2000 image compression are implemented depend on the traditional definition both on CPU and GPGPU. Therefore, the image compression quality is basically the same. And the test time of the experimental results on the CPU and GPGPU is shown in Table1.It can be seen from the experimental results that the speedup compared with the CPU test results which has not optimized is more than three times for those images with small data to process relatively. And for those images with large data to process is more than 15 times. Visible, the computing time of DWT algorithm in JPEG2000 which is optimized by CUDA on GPGPU is much less than the time on CPU. And with the increasing of the amount of data to calculate, the computing time of CPU showed significant growth trend, but the computing time of GPGPU grew slower. It can be seen from the speedup that with the increasing of the amount of data to calculate, the GPGPU has the superior performance for accelerating calculation.

VI. CONCLUSION

Implementing JPEG2000 image compression algorithm on CPU and GPGPU respectively can be concluded that in the field of scientific computing, especially for the large-scale intensive floating-point data which was not significantly related like image compression, using CUDA platform on GPGPU implement algorithm can obtain superior performance. Although parallel speed-up calculation on GPGPU have some restrictions by technical difficulties, such as compatibility, algorithm transplanting and optimization. Parallel computing on GPGPU has great potential and unparalleled advantages about performance which has been obtained the worldwide attention in different industries.Parallel accelerated computing on GPGPU will become the direction of the future development.

VII. REFERENCES

[1] Jing Guo, Qingkui Chen:Computer Engineering and Design,In Chinese,Vol.31(2010),p.3302-3304

[2] John D.Owens, David Luebke, Naga Govindaraju:Computer graphics forum,Vol.26(2007),p.80-113

[3] Han,T.D.,Abdelrahman,T.S:IEEE Transactions,Vol.2(2011),p.78-90

[4] Xiaoli Song, Qing Wang: Computer Measurement & Control,In Chinese,Vol.17(2009),p.1169-1171

[5] Joaquín Franco, Gregorio Bernabé:IEEE Computer Society,Vol.40(2009),p.111-118

[6] Sunpyo Hong,Hyesoon Kim:ACM SIGARCH Computer Architecture News,Vol.37(2009),p.152-163