Asynchronous Parallel Computing Model of Global Motion Estimation with CUDA

Zuo Chen
School of Computer and Communication, Hunan University, Chang sha, China
Email: chenzuo@hnu.edu.cn

Jialiang Ji, Renfa Li
School of Computer and Communication, Hunan University, Chang sha, China
Email: {jijialiang@hnu.edu.cn, lirenfa@vip.sina.com}

***Abstract*—For video coding, weighing the balance between and coding rate image quality, we apply global motion search algorithm to avoid loss of image quality and parallel computing capacity of graphics processors to accelerate the encoding process. According to the heterogeneous system of CPU+GPU, and the multi-threaded parallel structure, thread synchronization features of CUDA platform, we build a proper global motion search on CUDA computing model; taking CUDA thread synchronization mechanism to solve the problem of data consistency and improve the efficiency of on-chip data communication; taking CUDA asynchronous mechanism to hide the delay caused by the CPU functions. Demonstrated by the experimental results, parallel computing model based on CUDA could significantly improve the efficiency of motion estimation algorithm and a certain improvement gains from the asynchronous parallel model based on CUDA asynchronous system.**

***Index Terms*—CUDA, Computational model, Motion estimation, Video Coding, Asynchronous mechanism**

I. INTRODUCTION

With the development of network and communication, the multimedia service is becoming one of the most popular applications. Under the constraints of bandwidth, there is a great need for the high compression technology of video-audio data, such as video conferencing, digital storage media, television broadcasting, Internet streaming media and communications applications.

Reference[1] depending on the characteristics of high compression ratio, low bit rate transmission and network robustness, some of the video coding technologies have become industry standards that are widely used, such as H.264, MPEG-4 and AVS.

However， the algorithms from these video coding standards are complex and time-consuming. Therefore, the image quality tends to be reduced to improve coding efficiency in applications. Reference[2][3] the research on how to make a balance between the coding rate and the encoding quality has become the hot spots in relevant areas of IT.

Sponsored by Hunan science and technology project (2010gk2002), China.

Reference[4][5] motion estimation is an essential component of high compression rate video coding standard, which is the critical part of stable quality of sports image and high-quality and high compression ratio of time domain frame sequence. Block matching of motion estimation search algorithm is the prime operation, with the most complexity and the large part of computation in video coding. Provided the guaranteed image quality, how to improve the efficiency of motion estimation algorithm is the focus of research in video compression coding technology.

In recent years, the frequency and parallel degree of graphics process unit were enhanced significantly. It got a high computing performance in some applications. GPU-based general-purpose computation (GPGPU: General-Purpose Computing on Graphics Processing Units), has become a new field. Its major study is how to make use of GPU to carry out more kinds of general computing, other than process graphic data. Reference[6][7] as one of the best graphic card manufacturers in the world, NVIDIA has introduced a similar syntax CUDA-C programming model to support general purpose computation of graphics processors, providing a convenient GPGPU research and development platform.

Reference[8][9] the applications of video coding based on GPU have its natural advantage. Many researchers have attempted to apply video compression in CUDA environment. However, these studies show the performance of video coding efficiency could be improved significantly. Relevant researches are only limited in the multi-threading model of CUDA, but the time-consuming problem of communication between board memory of GPU and the main memory has not been resolved and hindered the further acceleration of video encoding.

Based on heterogeneous computing systems of CPU + GPU, this paper builds a parallel computing model for the global motion estimation algorithm. In order to make further improvement, it makes use of asynchronous concurrency mechanism of CUDA micro-architecture to hide time-consuming which is caused by memory communication. At last, we compared the acceleration effects of motion estimation in different computing architecture.

doi:10.4304/jcp.7.2.341-348

Ⅱ. MOTION ESTIMATION

In predictive coding of video frame, the active image frames in a scene have a certain correlation. The basic idea of motion estimation is to divide the active frame into many non-overlapping macro blocks, and assume that all pixels in the macro block have the same displacement; then, to find the most similar macro blocks in a search window of the reference frame for each current macro block in current frame, which is the matched block. The relative displacement of the matched block and the current block is motion vector. As shown in Fig.1, the current frame of a video sequence is divided into M x N non-overlapping blocks, on the assumption that all the pixels of each block have the same motion displacement.

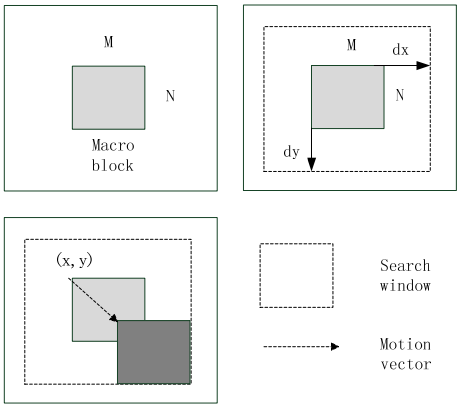


Figure 1. Motion estimation search.

Motion vector of each block is displacement between the current macro block and the best matched block in a limited search windows P x P of the previous frame. Matching algorithms focus on the error cost function, which usually uses SAD -the sum of the absolute value of difference between two matched blocks. If a (i, j) and b (i, j) is the pixels of current block and the matched block; m and n is the coordinates of the movement vector (MV), which is the location of the match block in the searching window. SAD can be defined in (1) (2) as follows:

$$SAD(m,n)=\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|a(i,j)-b(i+m,j+n)|. \quad (1)$$

$$-p \le m,n \le p-1. \quad (2)$$

Comparing current block with all matched blocks in the search windows and getting the minimum of SAD as the prediction of current block: Motion vector formula (3) (4) defined as follows:

$$MV=(MV_x,MV_y)=(m,n)\,|\,SAD_{\min}. \quad (3)$$

$$SAD_{\min}=\min[SAD(m,n)]. \quad (4)$$

Compared with several possible searching algorithms, the full search method can match all blocks in the search window and get the optimal matched block. According to the matching criterion of SAD, the global motion estimation algorithm has to calculate (1 +2 * dx) * (1 +2 * dy) SADs (Fig. 1: dx, dy is the margin of the macro block and the search window) in a coding frame. But compared with other motion search algorithm, it could get the global optimal motion vector. Based on global motion estimation algorithm, the search path shown in Fig. 2.

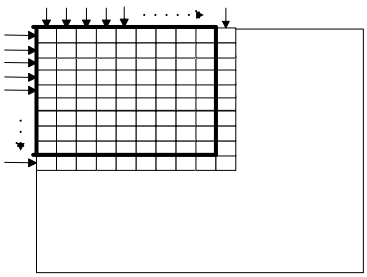


Figure 2. Mobile of starting search point.

Solid down-arrow shows the searching process of the matched block; the solid right-arrow shows the searching process of the matched block on vertical movement. In the matching process of adjacent blocks, reading data is independent to the neighboring points in the serial algorithm, so that the total reading-time increased because of the sum of each reading.

In the CUDA architecture, this data-correlation structure is conducive for multiple threads in groups to read frame data: firstly, neighborhood of data in a block makes data-accessing efficiency; secondly it gets high utilization of data and less invalid of reading data from memory. The hierarchy structure of motion searching process is consistent with the multi-thread architecture of CUDA framework. With adjacent nature of the pixel, the points in motion-searching can be combined into groups to make data-accessing more efficient. Therefore, in theory, CUDA architecture can effectively accelerate the motion estimation

Ⅲ. THE PROCESSING MODEL OF CUDA

A. The Physical Structure Of CUDA

Reference[10] architecture of CUDA is to set a scalable multi threaded Stream Processor (SM) array as the center. As the host CPU program calls the kernel on the CUDA grid, the grid blocks will be enumerated and distributed to the implementation of capacity with available multi-processor. Thread in a thread block execute concurrently on a multiprocessor. At the termination of a thread block, it will start a new block on free multi-processor. Multi-processor contains eight scalar processor (SP) cores, two for a priori (transcendental) the special function unit, a command unit, and chip multi-threaded shared memory. Multi-processor hardware to create, manage and implement concurrent threads, and scheduling overhead keep zero. It can be achieved barrier synchronization through an internal instruction _syncthreads()_. Fast barrier synchronization and lightweight thread creation and zero-overhead thread scheduling combined it effectively provide support for fine-grained parallelism. Shown in

Figure 3, each multi processor has one memory chips of the four following types;

- Each processor has a local 32-bit register;
- parallel data cache or shared memory, by all scalar processor cores sharing, shared memory space is located here;
- cache read-only fixed by the scalar processor cores all share, to accelerate from a fixed memory space read operation (this is a memory device read-only area);
- a read-only texture cache, the core of all scalar processor sharing, speed up the memory accessing of the texture read operation, each multi-processor will be addressed by implementing different models and data filtering unit to access the texture cache texture.

Local and global memory space which is device memory read/write area should not be cached.

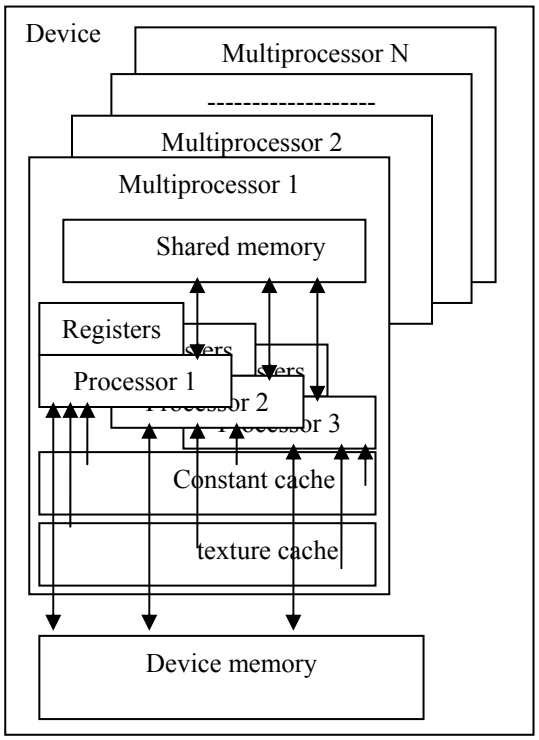


Figure 3. memory and processor model on Device

With a group of shared memory chip multiprocessor, SIMT unit will choose a ready warp block to execute one instruction and the next instruction is sent to the activities threads of the warp. Each warp execute a common block of instruction; so when the warp threads block is recognized by all 32 of its execution path, you can achieve maximum efficiency. If a thread warp, independent from the data, scattered by conditional branches, warp block will execute in the continuous implementation of the various branches, but not disable the thread on this path; on completing all paths, the thread back merged to the same executing path. Branches occurred only in the warp, a different warp block always execute independently, whether they are executing a common code path or another independent code paths.

B . Programming Model Of CUDA

Reference[10] CUDA provides a programming model that is ANSI C, extended with several keywords and constructs. The programmer writes a single source program that contains both the host (CPU) code and the device (GPU) code. These two parts will be automatically separated and compiled by the CUDA compiler tool.

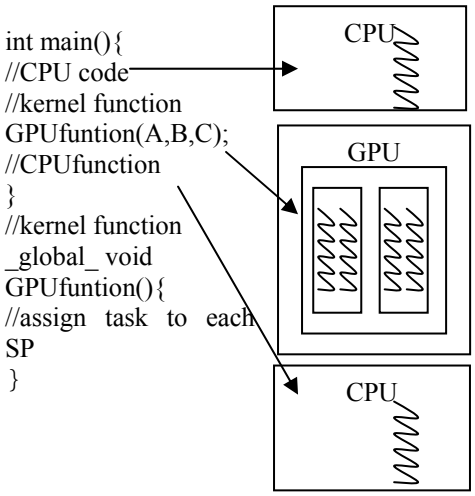


Figure 4. thread process on CUDA

CUDA allows the programmer to write device code in C functions called kernels. A kernel is different from a regular function in that it is executed by many GPU threads in a Single Instruction Multiple Data (SIMD) fashion. Each thread executes the entire kernel once. Launching a kernel for GPU execution is similar to calling the kernel function, except that the programmer needs to specify the space of GPU threads that execute it, called a grid. A grid contains multiple thread blocks, organized in a two-dimensional space (or one-dimensional if the size of the second dimension is 1). Each block contains multiple threads, organized in a three-dimensional space.

Meanwhile, the synchronization mechanism provided by CUDA, host code and device code can be executed in parallel, and maintain consistency. So it is necessary to match the distribution of tasks between the two, making the CPU GPU, heterogeneous systems to obtain high-performance computing power.

C. Data Model On CUDA

CUDA architecture matched the characteristics of multiple processors; each processor must allocate the appropriate amount of computation parallel. When the calculation is completed, the synchronization mechanism provided by CUDA achieves results in the merger process to ensure consistency in the middle of processing data. As shown in figure 5, the original data must be divided; each block was obtained after SP treatment results of a parallel.

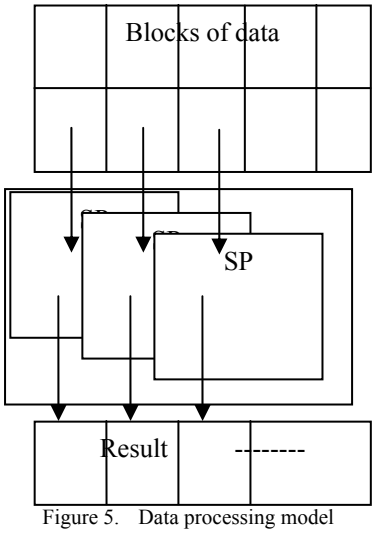


Figure 5. Data processing model

The result could be the final result returning to host the CPU or the intermediate results which will be calculated on GPU again according to the characteristic of the result then returning the final results back to the host.

So, when designing the computing model, we should analysis the characteristic of the original data and the parallel-ability of the computation flow.

Ⅳ. GLOBAL MOTION ESTIMATION COMPUTATION MODEL WITH CUDA

CUDA supports a large number of thread-level parallelisms, schedules and executes these threads concurrently in hardware dynamically. The main idea of CUDA programming model is to take the CPU as the host, while the GPU as a co-processor or device, so that GPU can process highly parallel tasks. In other word, it makes GPU and CPU working collaboratively. GPU is good at processing highly parallel data; while CPU is fit for dealing with other non-parallel computing tasks. Reference[6][7] in order to maximize the performance of applications based on CUDA, it needs to make full use of CUDA characteristic such as the organized characteristics of threads、asynchronous / synchronous mechanism、memory differences and so on, and build parallel computing model based on the characteristics of the applications.

A. Parallel Computing Model

In the CUDA architecture, the smallest execution unit of graphics chip is thread. Multi threads could be consisted to be a block. Reference[8][9] he threads in a block can exchange data in shared memory by thread synchronization mechanism. The Blocks which execute the same procedure formed a Grid. Block can not communicate with each other and there is no specific order of execution. These are the key features of CUDA: Thread is organized into two levels and communicate at lower levels through shared memory and thread synchronization. The hierarchy relations of CUDA architecture thread, as shown in Fig. 6.

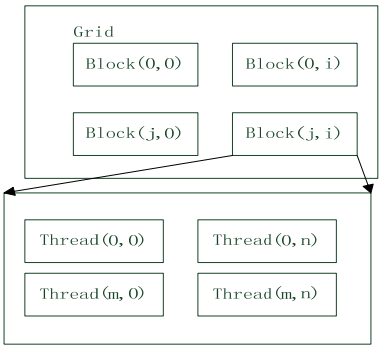


Figure 6. Multi-threaded architecture of CUDA.

In traditional parallel accelerated algorithms, the matching process is completed in one node and accelerated by the nodes' parallel computing ability because of the strong data correlation in matching process. Fig. 1, the matching process of each block has a certain mutual independence, so it has the parallel computing capabilities. Reference[7][10] because there were no communications between blocks, with the relevance of merged-accessing memory, matching process must be completed in one block. One frame needs NUM compares. The formula of NUM is as follows:

$$NUM = (2d_x + 1) * (2d_y + 1). \tag{5}$$

Therefore, it requires NUM threads in one Block. Reference[10] According to the parallel computing model shown in Fig. 7, the tasks of each Block are executed in different SM processor parallel and independently.

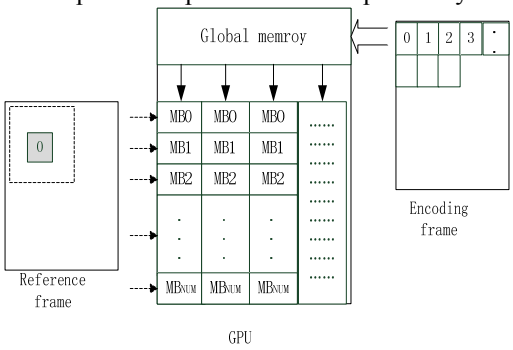


Figure 7. A macro-parallel model.

After being transferred into global memory of GPU, encoding frame was allocated to each Block and the reference frame was loaded too. Then, each block of encoding frame was matched parallel with blocks in search windows of reference frame until getting the optimal block and SAD circularly in each Block. In the process of motion estimation, each macro block can not communicate with each other and could be executed independently. Reference[10] as shown in Fig. 2, in each matching cycle of the macro block, the data is read at adjacent memory unit and there are much shared data among blocks.

Video data are organized by 16 * 16 macro block. In the device memory, merged-accessing memory can improve the efficiency of data accessing; as shown in Fig.

8, a thread group containing 16 threads can merged-access 16 memory units.

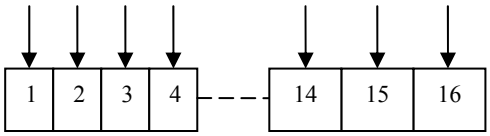
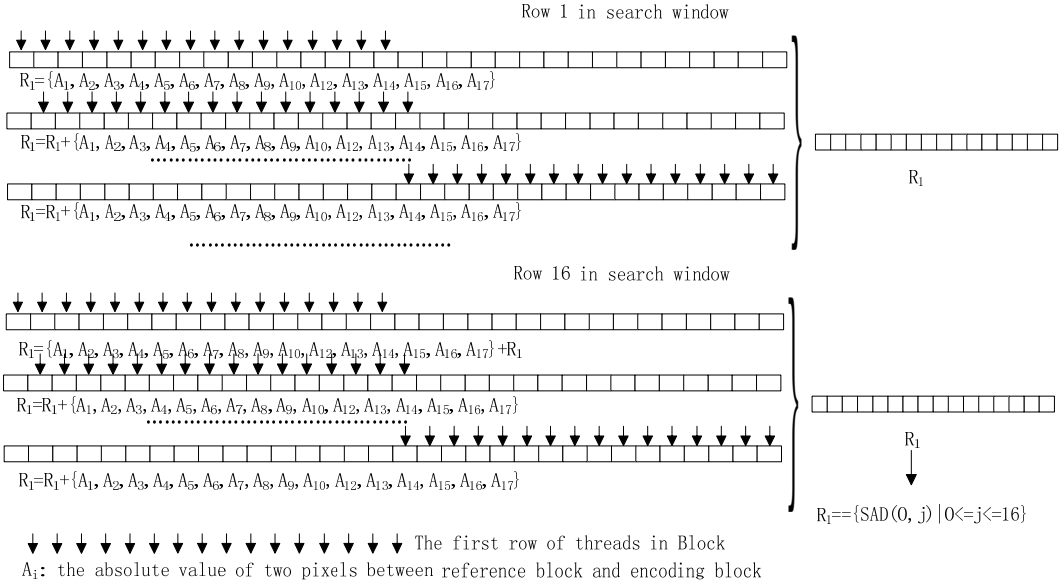


Figure 8. Merged-thread accessing.

Reference[10] at the micro level, as shown in Fig. 9: the memory unit, on merged-accessing of grouped-thread, can be read in one clock period or two. In Blocks, each thread in each loop can get the value, such as defined in (6).

$$A(m,n) = \left| a(i,j) - b(i+m, j+n) \right| . \quad (6)$$

$A(m,n)$ is the absolute value of pixel between the current encoding macro block and the matched macro block in search windows; As shown in (6), (i, j) is the index of pixel in encoding frame and (m, n) is the index of macro block in search windows. It was computed parallel by threads in one Block, while threads in one can execute parallel, so we can get a matrix as $\{A(m,n)\}$ in one cycle. This process can be described in a micro parallel computing model defined in Fig. 9.



(a) Executing the first row of threads in Block

$$\begin{Bmatrix} SAD(0,0) & SAD(0,1) & \dots & SAD(0,16) \\ SAD(1,1) & & & \\ \dots & & & \\ SAD(16,0) & SAD(16,1) & \dots & SAD(16,16) \end{Bmatrix} = \begin{Bmatrix} R_1 \\ R_2 \\ \dots \\ R_{17} \end{Bmatrix} = \sum_{1<=i<=16*16}^{i} \begin{Bmatrix} A(0,0)i & A(0,1)_i & \dots & A(0,16)_i \\ A(1,1)_i & & & \\ \dots & & & \\ A(16,0)_i & A(16,1)i & \dots & A(16,16)_i \end{Bmatrix}$$

(b)The threads in a Block execute parallel

Figure 9. Parallel computing model of a macro block.

As shown in Fig. 9(a), the first row of threads in Block, execute parallel at micro level and accumulate {Ai} parallel by merged-threads. At the end of cycle, we can get the vector-{SAD (i, j)|0<=j<=17}. While on CUDA platform shown in Fig. 7, at the macro level, the matching-process execute parallel by MB unit; and all threads execute and accumulate the matrix (shown in Fig. 9(b), the right part of the equation) parallel. The parallel computing model is suitable for synchronous execution on the Block. At the micro level shown in Fig 9, the merged-threads can improve data access efficiency. The model makes full use of merged-accessing mechanism as 'warp' (key word in CUDA) to improve efficiency of data-accessing.

Synchronous computing model based on multi-thread of CUDA platform, processes all macro blocks parallel in one coding frame by Blocks, and all threads in each Block process the matched reference blocks in parallel. Each processor of CPU or CUDA can implement parallel computing, but can not execute at the same time; it should start one task after another completed. So we should synchronize serial data-accessing to guarantee consistency.

B. Asynchronous Computing Model

In the above section, we built a parallel computing model based on CUDA at the macro level and micro-level; making use of highly parallel GPU computing

power for motion search algorithm to speed up the video compression coding. The computing model makes full use of the multithreaded nature of CUDA architecture. This section will build a parallel model on asynchronous concurrency of CPU+GPU's heterogeneous systems based on the model above. The model can be described the algorithm as a flow chart, shown in Fig.10.

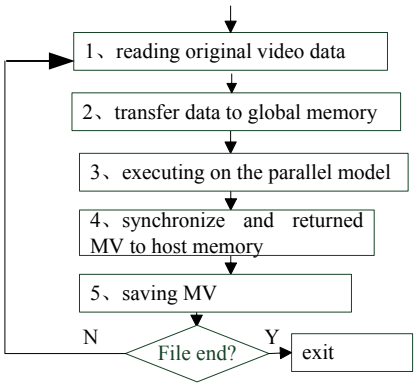


Figure 10. Synchronous parallel computing model.

In the parallel computing model, it needs to ensure data consistency. As shown in step 4, when using CUDA parallel model for motion search, we need to synchronize threads on CUDA to ensure that all of threads are finished. In this computing model, CPU and GPU should work serially one by one. According to the asynchronous nature of micro architecture of CUDA, the main thread will return control to the host after starting this parallel computing model of step 3. That is, CPU could perform other tasks at the same time.

As asynchronous characteristics of CUDA, the heterogeneous systems of CPU + GPU can hide part of CPU execution time effectively to improve the program's performance greatly, especially when the CPU execution time is similar to GPU execution time. Based on the parallel model in the last section, an asynchronous parallel model can be built with CUDA asynchronous mechanism.

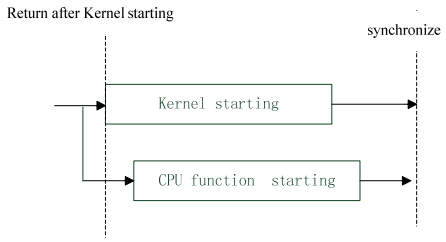


Figure 11. Asynchronous concurrency mechanism.

Shown in Fig. 11, on starting GPU function, the control of main thread immediately returned to the CPU function, similarly starting at the same time. Executing motion estimation of GPU in Step 3 is weakly consistent with reading data from hard disk of CPU at step 1, so the process of reading hard disk can be activated while the kernel is executing in GPU, in other word, reading data and motion search can be executed asynchronous and concurrently.

In the theory of asynchronous concurrent mechanism, if the time of the GPU implementation is more than that of CPU execution, then the CPU execution time can be effectively hidden; other cases, it can hide some delay caused by CPU functions. As a whole, it can improve acceleration efficiency of motion search. According to program flow shown in Fig. 10, combined with asynchronous shown in Fig. 11, the asynchronous parallel computing model can be built as shown in Fig. 12.

Shown in Fig. 12, after the init reading data from hard drive and transmit data to the GPU; this model then starts the kernel function of parallel model which is described in Fig. 7 and Fig. 9. While kernel function releases the main control of program, the CPU function immediately starts as Step 4. Concurrent execution shown in Step 3 and Step 4 hide the time of reading the hard disk by kernel function.

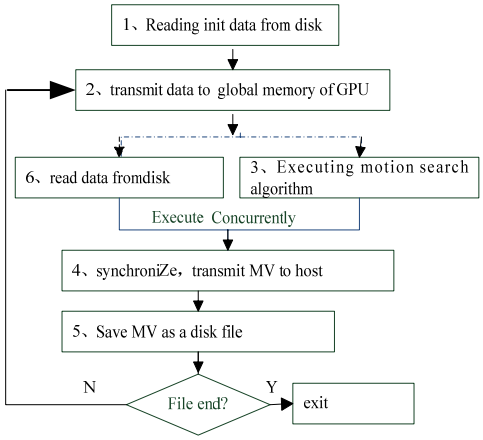


Figure 12. Asynchronous parallel computing model.

V. ANALYSES EXPERIMENTAL

In order to analysis the performance of the parallel model, the selected development environment is: Intel Pentium (R) Dual-Core CPU E5200 2.5GHz; Memory: 1G; NVIDIA Geforce GT220; windows xp sp3; Microsoft visual studio 2005; CUDA Toolkit2.3 and sdk2.3; NVIDIA CUDA driver supported in XP platform.

To avoid the unstable of operating environment factors in execution, we test several times for each case and select smallest time-consuming and repetitive results. With two orders of magnitude on the number of frames, we compared the encoding frame rate and deduced the relationship in encoding process between frame rate and frame number. As shown in table 1, it is the experiment results, separately based on the CPU and GPU. The speedup (6) is the comparison on time-consuming (ms) or frame rate (fps), which could reflect the accelerating effect of computing model with CUDA.

$$speedup = \frac{(\text{GPUcoding rate : fps - CPUcoding rate : fps})}{\text{CPUcoding rate : fps}} \quad (6)$$

TABLE I.
COMPARISON OF EXPERIMENTAL DATA

resolution	Num of frame	fps		
		CPU	GPU(sync)	GPU(Async)
176*144	600	123.54097	651.1646443	669.5453789
	2000	127.87047	649.5443422	671.7346855
352*288	600	31.462383	201.1264983	213.5527854
	2000	34.184675	199.6575893	212.6100389
704*576	600	6.809567	45.43790246	52.06548458
	2000	6.956448	49.53347983	56.93452977

As shown in TABLE Ⅰ, at the same resolution, the coding rate of frame is similar to other which encoded in different orders of magnitude by the same computing model. It indicated the weak association between encoding rate and the number of frame. While encoding on different resolution by the same computing model, the decreasing of encoding rate indicate a great association between coding rate and video resolution. In Fig. 13, it is the comparison chart of coding frame rate for both platforms.

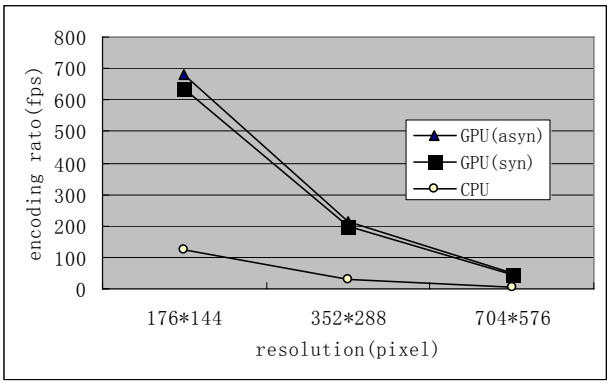


Figure 13. Comparison of Asynchronous Experiment (The Case of 2000 frames).

The horizontal axis shown in Fig. 13 represents the three different resolutions in Table 1, and the vertical axis represents the coding rate of frames. As the resolution increases, coding rate decreased and the curves were of the same trend. As shown in Fig. 13, compared to CPU-based algorithms, the encoding frame rate have been significantly improved which was accelerated by micro-architecture GPU based on CUDA. The number of coded blocks per frame increased as the resolution increasing, so that it required much more Blocks. It resulted in an overall increase of coding time and reduction of coding rate.

Concluding form the comparison in Fig. 13, under the asynchronous mechanism, coding rate of asynchronous model is always higher than the synchronous experiment. Evolved from the speedup formula, the calculation method shown in (7) we can get the improvement of performance based on asynchronous computing model.

$$\text{added efficiency\%} = \frac{\text{async speeddup - sync speedup}}{\text{sync speedup}} \quad (7)$$

TABLE II.
THE IMPROVEMENT BASED ON PERFORMANCE DATA FOR THE SPEEDUP

resolution	Num of frame	speedup		Added efficiency%
		GPU(sync)	GPU(Async)	
176*144	600	4.27083966	4.419622162	3.483682688
	2000	4.079705597	4.253243266	4.253681176
352*288	600	5.392602185	5.787559143	7.324051452
	2000	4.840558358	5.219454738	7.827534588
704*576	600	5.67265664	6.645931757	17.15730704
	2000	6.120513203	7.184425409	17.38272872

As shown in TABLE Ⅱ, the speedup of asynchronous mechanism is higher than the synchronous computing model. As seen from the added performance of asynchronous mechanism, there is a certain improvement by the optimization of the asynchronous mechanism, increased about 3.4% to 17.4%. When the frame resolution gets higher, it will take longer time to read and write on the hard drive and much more time can be hidden by asynchronous mechanism. As shown in TABLE Ⅱ, it got approximately 17.4% improvement on the accelerated coding at 704*576, almost 2-5 times to the low-resolution case.

From the analysis of experimental data above, it shows that the parallel computing model of motion estimation has get a significant effect on the encoding acceleration based on the double-parallel architecture of multi-threaded on CUDA; and the parallel computing model could get a further improvement based on the optimization of asynchronous mechanism on CUDA,.

Ⅵ. SUMMARY

This paper analyzes the principles of global motion estimation algorithm and the feasibility of parallel computing in the video encoding process. It minutely describes the heterogeneous architecture of GPU + CPU and builds a parallel computing model which combined with CUDA platform around the search algorithm of global motion estimation. According to the asynchronous and concurrent mechanisms of multi-threads on CUDA, the computing model gets a further optimization and improvement. At the same time, we realized this model and experiment for several cases. At last, we optimize this model by asynchronous mechanism and experiment for the same cases too. The results show that GPU as a co-processor of CPU can effectively accelerate data-intensive and computing-intensive calculations. Any more, the optimization based on the asynchronous mechanism can get further improvement.

REFERENCES

[1] Joint Video Team of ITU-T and ISO/IEC JTC 1. Advanced video coding for generic audiovisual services [M], 2005, pp.1-3

[2] Xing Xianglei, Du Sidan. Parallel Implement of the H.264 Video Encoder Based on Cluster of Workstation. Journal of southeast university (natural science edition),Vol38, No.6, Nov 2008

[3] Li Ming, Hu Ruimin, Li Wei. Parallelization of vdieo coding algorithm. Computer Engineering and Applications. No.16, 2003, pp.109-112.
[4] Ning hua, Mei Zheng, Li Jintao. Slice- based parallel algorithm of h.264 video encoder. Computer Engineering. Vol.31, No.4, pp.181-182,211, February 2005
[5] Jiang Xingchang, Zhou Jun and Luo Chuanfei. H.264 research of parallel coding algorithm in H.264 [J].video engineering. No.02,Vol32, 2008
[6] Reiji Suda, Takayuki Aoki, Shoichi Hirasawa. Aspects of GPU for General Purpose High Performance Computing. Design Automation Conference, 2009. ASP-DAC 2009. Asia and South Pacific.19-22,pp.216-223, Jan, 2009
[7] Ziyi Liu, Wenjing Ma. Exploiting Computing Power on Graphics Processing Unit. Computer Science and Software Engineering.2008 International Conference on. Volume 2, 12-14, pp.1062 -1065, Dec, 2008
[8] Wei-Nien Chen, Hsueh-Ming Hang. H.264/AVC MOTION ESTIMATION IMPLMENTA- TION ON COMPUTE UNIFIED DEVICE ARCHITECTURE (CUDA), Multimedia and Expo. 2008 IEEE International Conference on June 23 2008, pp.697-700,April,26,2008
[9] Seung In Park, Sean P. Ponce, Jing Huang, Yong Cao and Francis Quek. Low-Cost, High-Speed Computer Vision Using NVIDIA's CUDA Architecture. Applied Imagery Pattern Recognition Workshop, 2008. AIPR '08. 37th IEEE 15-17,pp.1-7 Oct 2008
[10] NVIDIA corporation, *NVIDIA CUDA Compute Unified Device Architecture- Programming Guide* Version 1.1. 2008.