Fast fractional-pel interpolation algorithm of H.264 based on CUDA

X G Hong[1], H Liu[1*], Y Xiao[2]

[1]*School of Information Engineering, Nanchang University, Nanchang 330031, China*

[2]*Institute of Computing Technology, Nanchang 330031, China*

Received 12 June 2014, www.tsi.lv

Abstract

H.264 video standard introduces fractional pixel motion compensation technology to obtain a more precise motion vector and a higher compression ratio. But, it increases the complexity of the motion compensation process at the same time. In order to solve the difficulties, we analysis the procedure of fractional-pel interpolation in H.264 and propose a fast fractional-pel interpolation algorithm based on CUDA. Experimental results show that the fast algorithm enables locating fractional pixel effectively and improves the speed of fractional pixel motion estimation. Compared with the CPU serial algorithm, the fast algorithm can significantly improve encoding rate almost four times in processing high-resolution video sequences.

Keywords: H.264, CUDA, interpolation algorithm, fractional-pel

1 Introduction

H.264 is a highly compressed digital video codec standard. It is proposed by the *Joint Video Team* (JVT), which is grouped by the *Video Coding Experts Group* (VCEG) of ITU-T and the *Moving Picture Experts Group* (MPEG) of ISO/IEC [1]. Compared with other video coding standards such as MPEG-2, H.263, the H.264 standard has high compression ratio and high adaptability to the network. Because of that feature, the H.264 standard can be widely used in digital television, wireless video communication, video conference over IP and other multimedia services.

Nevertheless, the design of H.264 uses intra prediction in intra-frame, multiple frames reference capability, quarter-pixel interpolation, and flexible macro-block ordering in order to enhance *Motion Estimation* (ME) and *Motion Compensation* (MC). But, it also increases the complexity of algorithm and the encoding computation. By analysing the H.264 encoding process, we conclude that the inter prediction takes more than 75% of the encoding time [2]. Moreover, in the process of inter prediction to get the fractional pixel has cost most of the calculation. So, in order to improve the computing speed of inter prediction, we must take an efficiently and fast interpolation algorithm to get the fractional pixel.

On the other hand, personal computers commonly equipped with GPU. Recently, the progress of GPU has caught a lot of attention; they have changed from fixed pipelines to programmable pipelines; the hardware design also includes multiple cores, bigger memory sizes and better interconnection networks which offer practical and acceptable solutions for speeding both graphics and non-graphics applications. GPU are highly parallel and are normally used as a coprocessor to assist the Central Processing Unit (CPU) in computing massive data.

NVIDIA developed a powerful GPU architecture denominated Compute Unified Device Architecture (CUDA), which is formed by a single program multiple data-computing device. Hence, the fractional pixel interpolation algorithm developed in the H.264 encoding algorithm fits well in the GPU philosophy and offers a new challenge for the GPU.

This paper proposes a fast fractional pixel interpolation algorithm to accelerate the half-pixel and quarter-pixel in the process of inter prediction in the H.264 by using CUDA. The proposed algorithm efficiently develops the parallel computing of GPU to implement the parallel computing of fractional pixel interpolation. The remainder of this article is organized as follows: Section 2 briefly introduces fractional pixel interpolation process in H.264. Section 3 the fast fractional-pel interpolation algorithm processing based on CUDA is presented. Experimental results and compared to traditional approach are discussed in Section 4. Sections 5 show the conclusion and some ongoing work.

2 Overview of H.264 fractional-pel interpolation process

H.264 uses the 1/4 pixel precision to complete Motion Estimation (ME) and Motion Compensation (MC). Compared with 1/2 pixel precision in H.263, it can get more than 2dB coding gain [3].The mainly process of fractional pixel interpolation algorithm shows as follows [4]:

I. Get the half-pixel: The half-pixel where between two integer-pixels (as b, h, m, s, etc. shown in Figure 1 (a)).

* ***Corresponding author*** e-mail: liuhao_ncu@sina.com

H.264 uses A sixth-order finite impulse response filter (FIR) to obtain the interpolated value by the neighbouring integer pixels. The weighted value of FIR is 1/32,-5/32, 5/8, 5/8, -5/32, 1/32.

We can get the half-pixel value of b as follows:

$$b = round((\mathrm{E} - 5\mathrm{F} + 20\mathrm{G} + 20H - 5I + J)/32), \quad (1)$$

Similarly, the integer-pixel include of A, C, G, M, R, T through the sixth-order FIR can obtain half-pixel value of h. Once all of the adjacent integer-pixels (vertical or horizontal direction) get their half pixels, the remaining half pixels such as j can be calculated by six of vertical or horizontal half pixels. For example, the half pixel of j is calculated by value of cc, dd, h, m, ee, ff, which is shown in the Figure 1 (a).

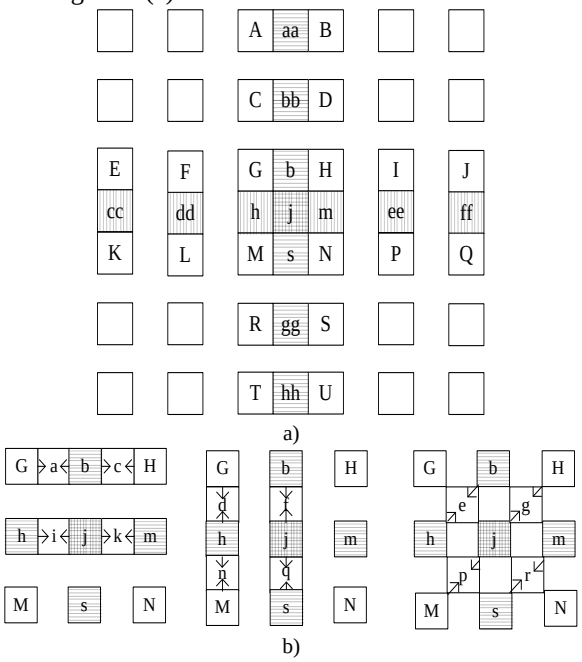

FIGURE 1 Fractional pixel distribution

II. Get the quarter-pixel: We use the linear Interpolation algorithm to obtain the value of quarter-pixel by the half pixel and integer pixel in the horizontal or vertical. The quarter-pixel distribution is shown in the Figure 1 (b). We can get the quarter-pixel value of a as follow:

$$a = round((\mathrm{G} + \mathrm{b})/2), \quad (2)$$

The remaining quarter-pixels such as e, g, p, r can be calculated by linear interpolation with a pair of half-pixel on the diagonal.

3 CUDA implementation of fractional-pel

3.1 CUDA PROGRAMMING MODEL

CPU and GPU work together in the CUDA programming model [5]. CPU is responsible for the serial parts of the code, and GPU is responsible for parts of the algorithm where is intensive and can be used to parallel Computing. The structure of the CUDA programming model is shown in the Figure 2. Typically, the different of the implementation of the algorithm in GPU and CPU is the kernel function. Kernel function is part of a parallel program, which is used to parallel computing. The kernel function is used *__global__* to declare function type in the CPU and executes on the device in the GPU.

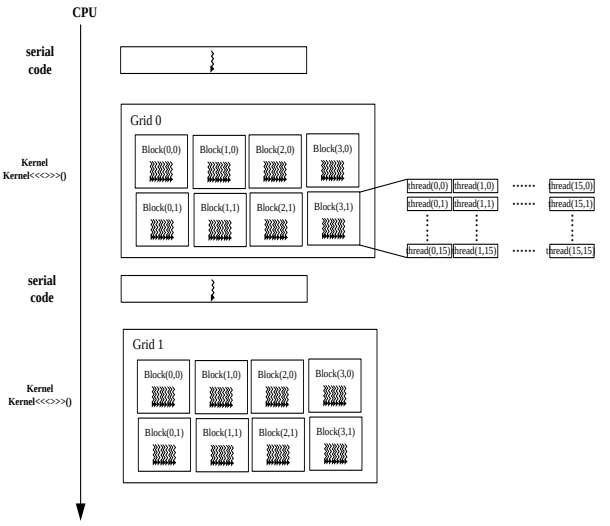

FIGURE 2 The structure of the CUDA programming model

3.2 ANYLYSIS OF FRACTIONAL PIXEL INTERPOLATION

By analysing the H.264 encoding process [6-8], we conclude that the 4×4 sub-block includes three kinds of possible pixel value such as integer-pixel, half-pixel and quarter-pixel. As shown in Figure 3, the red point mark as integer pixel such as the point of 0.The white point is the fractional pixels. Among the white point, the point of 2,8,10 is half-pixel, the others is quarter-pixel. According to the predicted locations can be divided into the following six kinds of situations in the process of inter prediction.

I. The prediction point right on the integer-pixel where is region of 0.In this case, we should not need to calculate other fractional pixel. This is the simplest type of situation.

II. The prediction point right on the region of 1_2_3.In this case, we must calculate the corresponding fractional pixel by the interpolation algorithm.

III. The prediction point right on the region of 4_8_12.This case is similar to the second case so that we use the same method get the fractional pixel.

IV. The prediction point right on the region of 6_10_14.In this case, the middle of half-pixel should use the other value of half-pixel to calculate. So, we must get the corresponding half- pixel first, then uses this half-pixel and the others half-pixel to get the value of quarter-pixel.

V. The prediction point right on the region of 9_10_11. This case is similar to the fourth case so that we use the same method get the fractional pixel.

VI. The prediction point right on the region of 5_7_13_15.In this case, all of the pixels is quarter-pixel

and they must be calculated by linear interpolation with a pair of corner half-pixel. The computation in this case is quite large because all of the quarter-pixel by linear interpolation.

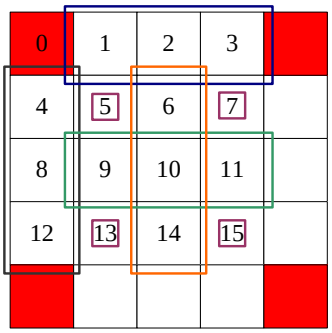

FIGURE 3 The 4×4 sub-block pixel distribution

From the analysis of the six cases, we can find that it needs according to different conditions to deal with the fractional pixel in H.264. In each condition it should use a large number of interpolations to derive the corresponding value of half-pixel and quarter-pixel. Because of this reason, the inter prediction occupies most of H.264 coding and decoding time. So, using the fast fractional-pel interpolation algorithm based on CUDA can accelerate fractional pixel positioning time so that it can improve encoding and decoding time and efficiency of H.264.

3.3 THE SPECIFIC IMPLEMENTATION PROCESS

Firstly, judging the region where the prediction point falls. Secondly, according to the prediction region get the corresponding half-pixel. Finally, according to the value of half-pixel get the corresponding quarter-pixel in the prediction region.

3.3.1 Judging the prediction region

In JM8.6 specification [9] defines a method named *get_block*(). In this method, there are two variables named dx and dy which are use to judge the location of prediction point. The variable of dx means the abscissa of the nearest integer-pixel on left. The variable of dy means the ordinate of the nearest integer-pixel on left. The specific conditions of the judgment are shown in the Table 1.

TABLE 1 Judgment of specific conditions

Judge conditions	Region of location
if (dx == 0 && dy == 0){}	region of 0
if (dy == 0){ if ((dx&1) == 1){….}}	region of *1_2_3*
if (dx == 0){ if ((dy&1) == 1){….}}	region of *4_8_12*
if (dx == 2){ if ((dy&1) == 1){….}}	region of *6_10_14*
if (dy == 2){ if ((dx&1) == 1){….}}	region of *9_10_11*
others	region of *5_7_13_15*

3.3.2 Calculate the fractional pixel in the corresponding region

In this step, we use the bilinear interpolation algorithm based on CUDA to get the value of half-pixel [10, 11]. Then parallel computing the corresponding quarter-pixel by the calculated half-pixel. We use the prediction point right on the region of 5_7_13_15 as an example to describe the fast interpolation algorithm based on CUDA to get the value of fractional pixel.

I. As shown in the Figure 1 (a), the half-pixel of b can be calculated by the four integer-pixels on the diagonal (C, D, M, N). Calculation method as follows:

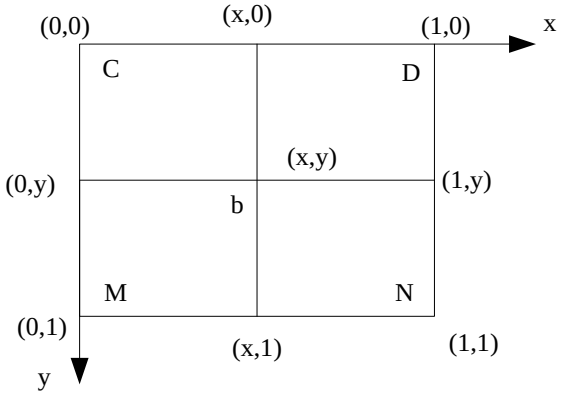

FIGURE 4 Bilinear interpolation algorithm

As shown in the Figure 4, the point of b is $F_{xy}(x,y)$, the value of integer-pixel{C: $F_{00}(0,0)$, D: $F_{10}(1,0)$, M: $F_{01}(0,1)$, N: $F_{11}(1,1)$ }is (f_{00}, f_{10}, f_{01}, f_{11}).

In the Y-direction, we use the linear interpolation the get the value of $F_{0y}(0,y)$:

$$f_{0y} = f_{00} + y(f_{01} - f_{00}). \tag{3}$$

In the Y-direction, we use the linear interpolation the get the value of $F_{1y}(1,y)$:

$$f_{1y} = f_{10} + y(f_{11} - f_{10}), \tag{4}$$

Then, we use the value of $F_{0y}(0,y)$ and $F_{1y}(1,y)$ to get the value of $F_{xy}(x,y)$ in the X-direction:

$$f_{xy} = f_{0y} + y(f_{1y} - f_{0y}), \tag{5}$$

Substituting Equations (3) and (4) into Equation (5), combination and simplification can get as follow:

$$f_{xy} = (1-x)(1-y)f_{00} + y(1-x)f_{01} + x(1-y)f_{10} + (xy)f_{11}. \tag{6}$$

Set: $v(x_0) = 1-x$; $v(y_0) = 1-y$; $v(x_1) = x$; $v(y_1) = y$ then $v_{00} = v(x_0)v(y_0)$; $v_{01} = v(x_0)v(y_1)$; $v_{10} = v(x_1)v(y_0)$; $v_{11} = v(x_1)v(y_1)$. Then, Equation (6) can be simplified as follows:

$$f_{xy} = v_{00}f_{00} + v_{01}f_{01} + v_{10}f_{10} + v_{11}f_{11}. \tag{7}$$

II. Using Equation (7) to calculate the half-pixel. The basic computational unit of fractional pixel in the H.264 is the 4×4 sub-block [12].We use a thread-block to calculate a 4×4 sub-block. For the resolution is W×H in a frame, the number of thread-block is shown as Equation (8):

$$n = [W/4][H/4]. \quad (8)$$

The number of half-pixel, which should be calculated in a 4×4 sub-block is 33. So, we should distribute 33 threads in a thread-block and each thread is responsible for calculating the value of a half-pixel. The flow diagram is shown in the Figure 5.

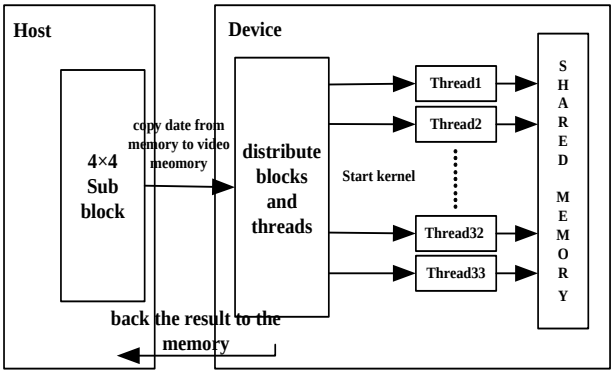

FIGURE 5 Flow diagram of calculate half-pixel based on CUDA

III. Calculate the quarter-pixel. There are two kinds of quarter-pixel that we should to calculate. One is used the integer-pixel and half-pixel by the linear interpolation to get (the point of a in the Figure 1 (b)). The other one is used a pair of half-pixel on the diagonal to get (the point of e in the Figure 1 (b)).The number of tread-block can be calculated by Equation (8).There are 120 quarter-pixels in a 4×4 sub-block, so, we distribute 120 threads in a thread-block and each thread is responsible for calculating the value of a quarter-pixel.

IV. The process of implementation on CUDA.

It can be seen from the CUDA programming model that the CPU serial code is responsible for data preparation and initialization of the device. The work which is shown as Host in the Figure 5 mainly includes memory allocation, video memory allocation, grid configuration, etc. The CPU will start the kernel function when the preparatory work is completed. The kernel function is used to parallel processing the data on the video memory by the GPU. To realize the algorithm proposed in this paper, we should create two kernel functions. One of the functions is used to calculate the half-pixel and the other one calculates the quarter-pixel. As the Device show in the Figure 5, the block in the kernel function creates threads to parallel computing [13] the value of half-pixel. Then, CPU executes its serial code to clean the kernel function and start next kernel function. At the same time, the CPU serial code will put the value of integer-pixel and half-pixel into video memory. After that, the kernel function will start the threads to parallel computing the value of quarter-pixel. The data will be copied from video memory to memory when fractional pixels have been calculated. Finally, the space of memory and video memory will be freed and exit the CUDA.

V. The pseudo-code, which describes implementation process is shown as follows:

```
Fractional_pel:
//allocation of video memory space
cudaMalloc()
//copy the data from memory to video memory
cudaMemcpy(cudaMemcpyHostToDevice)
dim3 block_half(n,1,1)
//calculate the number of thread, distribute 33 threads.
dim3 thread_half(33,1,1)
//obtain the value of half-pixel.
GPUhalf_pel<<<block_half,thread_half>>>()
//--------calculate the quarter pixel------
dim3 block_quarter(n,1,1)
//calculate the number of thread, distribute 120 threads. dim3
thread_quarter(120,1,1)
GPUquarter_pel<<<block_quarter,thread_quarter>>>()
//obtain the value of quarter-pixel
cudaMemcpy(cudaMemcpyDeviceToHost)
//free the space of memory and video memory
cudaFree()
```

4 Experimental results and analysis

In this work, we use NVIDIA GeForce 9600 GSO as GPU platform which presents the characteristics depicted in Table 2. The CPU platform is AMD Sempron™ Processor 3200+ 1.80GHz. We use Microsoft Visual Studio 2008 and CUDA5.0 as the programming platform which runs on the operating system of Window 7. We use *cuda_Nsiht_Visual_Stuio_Edtion* as the performance analysis tool.

TABLE 2 GPU main features

Characteristic	GeForce9600 GSO
Compute capability	1.1
Global memory	512M
Number of multiprocessors	6
Number of cores	48
Constant memory	64KB
Shared memory per block	16KB
Registers per block	8192
Active threads per multiprocessor	768
Max threads per block	512
GPU Clock rate	1.5GHz

The performance evaluation of the fast fractional-pel interpolation algorithm of H.264 based on CUDA that we proposed based on JM8.6 encoder [9]. The test sequences are shown in Table 3.

TABLE 3 Test sequences

Test sequences	Resolution	Frames	Sample format
Carphone	QCIF(176×144)	150	YUV 4:2:0
Akiyo	QCIF(176×144)	300	YUV 4:2:0
Bus	CIF(352×288)	150	YUV 4:2:0
Foreman	CIF(352×288)	300	YUV 4:2:0
Blue_sky	1080P	150	YUV 4:2:0
Riverbed	1080P	300	YUV 4:2:0

For testing, QCIF (176×144), CIF (352x288) and 1080p are selected. These three kinds of video sequences have different resolution and different number of frames. The sample format of them is YUV 4:2:0. Firstly, the

proposed algorithm is put into JM encoder to replace the original algorithm, and then we realize the proposed algorithm on CUDA-based GPU. At last, we integrate the CPU implementation with the original algorithm and the GPU implementation with the proposed algorithm in JM encoder. The test sequences shown in Table 2 are used to test these two implementations. We get the encoding rate of two implementations. Table 4 lists the results of the comparison results of CUDA technology and CPU implementations. And Figure 6 shows the improvement of these video sequences.

TABLE 4 Encoding rate comparisons

Test sequences	Encoding rate(fps)		Speed Up
	CPU	CUDA	
Carphone	8.93	9.34	1.05
Akiyo	8.34	9.25	1.10
Bus	5.72	8.67	1.52
Foreman	4.89	8.56	1.75
Blue_sky	2.56	8.45	3.30
Riverbed	2.23	8.33	3.74

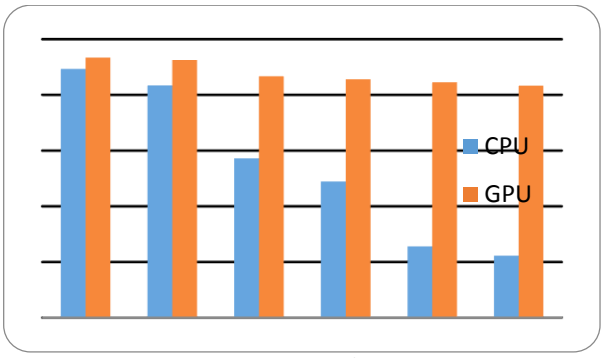

FIGURE 6 Comparison of encoding rate

From Table 4 and Figure 6, we can see the proposed algorithm for acceleration of the encoding rare in different resolution and the CUDA-based algorithm achieves higher efficiency than the CPU-based algorithm. For video sequences with the same resolution and different number of frames (*Carphone* and *Akiyo*, *Bus* and *Foreman*, *Blus_sky* and *Riverbed* shown in the Table 3), we can summarize that under the same resolution, the amount of encoding computation is not very large when video sequence frames is less. In this case, encoder takes most time in data copies when it uses CUDA so that the acceleration effect is not obvious. However, with the increase of the video frames and amount of computation, the advantage of the GPU parallel processing can be reflected and the speedup also increases.

For video sequences with the same number of frames and different resolution (*Carphone* and *Foreman*, *Akiyo* and *Blus_sky*, *Bus* and *Riverbed* shown in the Table 3), we can summarize that the acceleration effect of low-resolution video sequences is not obvious. However, with the increase of the resolution of the video, there are more 4×4 sub-block will be divided in a frame of the video sequence. In this case, the number of the fractional-pel, which should be calculated will increases so that it increases the amount of computation of the encoder and the encoding rate is also decreased.

However, from the speedup shown in the Table 4, we can know that the acceleration effect is more obvious with the increase of the resolution. This is also reflected that the fast algorithm based CUDA can significantly improve encoding rate in processing high-resolution video sequences.

5 Conclusions

In this paper, we take the advantage of advantage of parallel computing in CUDA and propose to use CUDA technology to speed up the H.264 fractional-pel interpolation. The results show it is an effective approach to deal with this highly data-adaptive processing algorithm and it is can use to deal with high-resolution video sequences.H.264 standard is the most widely used standard and it is important to optimize its algorithm and execution time on a continuous basis. For future work, we will continue to optimize other modules and reduce complexity of the whole process based on CUDA technology.

Acknowledgments

This project is supported by the Graduate innovation fund project of Jiangxi Province, China 2013.

References

[1] ITU-T RECOMMENDATION 2003 Advanced Video Coding for Generic Audiovisual Services *ISO/IEC* 14496
[2] Blasi S G, Peixoto E, Izquierdo E 2013 Enhanced Inter-Prediction Via Shifting Transformation in the H. 264AVC *Circuits and Systems for Video Technology* **23**(4) 735-740
[3] Wedi T, Musmann H G 2003 Motion and aliasing-compensated prediction for hybrid video coding *Circuits and Systems for Video Technology* **13**(7) 577-86
[4] Richardson I E 2004 H.264 and MPEG-4 video compression: video coding for next-generation multimedia *Wiley*
[5] NVIDIA 2012 CUDA Compute Unified Device Architecture Programming Guide Version 5.0 *Applications* http://developer.download.nvidia.com
[6] Fang Y, Zhou J 2006 Fast Fractional-pel Interpolation Algorithm of H.264 *Computer Engineering* (1) 076
[7] Chen Z, Xu J, He Y, Zheng J 2006 Fast integer-pel and fractional-pel motion estimation for H.264/AVC *Visual Communication and Image Representation* **17**(2) 264-290
[8] Wang Y J, Cheng C C, Chang T S 2007 A fast algorithm and its VLSI architecture for fractional motion estimation for H. 264/MPEG-4 AVC video coding *Circuits and Systems for Video Technology* **17**(5) 578-83
[9] Joint Video Team Software 2010 JM8.6 *Applications* http://iphome.hhi.de/suehring/tml/download/
[10] Gribbon K T, Bailey D G 2004 A novel approach to real-time bilinear interpolation *Field-Programmable Technology* 126-31
[11] Liu J J ,He Z, Chen L 2010 Bilinear interpolation of geomagnetic field *Computer Application and System Modeling (ICCASM)* V2-665
[12] Lu X, Tourapis A M, Yin P, Boyce J 2005 Fast mode decision and motion estimation for H.264 with a focus on MPEG-2/H. 264 transcoding *Circuits and Systems* 1246-9
[13] Farber R 2011 CUDA application design and development *Elsevier*

Authors

Xianggong Hong

Current position, grades: Associate professor, Nanchang University.
University studies: MS degree in School of Electronics and Communications Engineering at Nanchang University in 2009.
Scientific interest: image processing technology, video communication technology.

Hao Liu

Current position, grades: Master of Signal and Information Processing, Nanchang University.
University studies: Nanchang University in School of Information Engineering (2008-2015).
Scientific interest: communication and information system, video communication technology.

Yun Xiao

Current position, grades: natural science researcher of Institute of Computing Technology, Jiangxi.
University studies: MS degree at School of Computer Science from East China, Jiaotong University.
Scientific interest: data mining, database development.