멀티코어 시스템에서 쓰레드 수에 따른 병렬 색변환 성능 검증

김정길* 종신회원

A Performance Evaluation of Parallel Color Conversion based on the Thread Number on Multi-core Systems

Cheong Ghil Kim* *Lifelong Members*

요 약

멀티 코어 프로세서의 보급 확산으로 최근에는 임베디드 시스템에서도 채택되고 있다. 따라서 일반적으로 대규모의 컴퓨팅과 메모리 접근을 필요로 하는 멀티미디어 응용은 멀티 코어 플랫폼 기반의 병렬화가 가능하다. 본 논문에서는 멀티 코어 CPU을 이용한 효율적 색 공간 변환을 위한 스레드 수준 병렬 기법의 성능 향상을 검증하였다. 스레드 수준 병렬화 특히 멀티 코어 프로세서기반 공유 메모리 컴퓨팅 시스템에서는 매우 유용한 병렬 처리 패러다임이 되고 있다. 본 구현에서 스레드 수준 병렬화는 각 스레드에 다른 입력 픽셀을 할당하여 실행하였다. 성능 평가를 위해 직렬 및 병렬 구현들 사이의 처리 속도의 비교에 기초하여 대표적 멀티 코어 프로세서에서 색 변환을 위한 성능 향상 정도를 평가하였다. 결과는 스레드 수준의 병렬 구현에 관계없이 다른 멀티 코어에서 전반적으로 비슷한 성능 향상의 비율을 보여주었다.

Key Words : paralle processing, thread-level parallelism, multi-core processor, color conversion, performance evaluation, pre-processing.

ABSTRACT

With the increasing popularity of multi-core processors, they have been adopted even in embedded systems. Under this circumstance many multimedia applications can be parallelized on multi-core platforms because they usually require heavy computations and extensive memory accesses. This paper proposes an efficient thread-level parallel implementation for color space conversion on multi-core CPU. Thread-level parallelism has been becoming very useful parallel processing paradigm especially on shared memory computing systems. In this work, it is exploited by allocating different input pixels to each thread for concurrent loop executions. For the performance evaluation, this paper evaluate the performace improvements for color conversion on multi-core processors based on the processing speed comparison between its serial implementation and parallel ones. The results shows that thread-level parallel implementations show the overall similar ratios of performance improvements regardless of different multi-cores.

I. Introduction

With the rapid development of technology of CPU, multi-core processor architecture has emerged as a dominant market trend in desk-top PCs as well as embedded systems. This movement enables a single chip to increase the performance capability without requiring a complex system and increasing the power requirements [1,2].

In this situation, another solution to increase PC performance could be achieved by taking advantage of parallelism in software rather than depending on hardware technologies. Therefore, parallel programming becomes an important issue of multi-core systems [3,4] because it allows full use of the market dominant hardware systems. Especially, thread-level parallelism has been becoming

※ 본 논문은 2014년도 남서울대학교 교내연구비 지원에 의하여 연구되었음.
*남서울대학교 컴퓨터학과 (cgkim@nsu.ac.kr)
*접수일자 : 2014년 10월 23일, 수정완료일자 : 2014년 11월 10일, 최종게재확정일자 : 2014년 11월 17일

very useful parallel processing paradigm especially on shared memory computing systems with multi-core processors [5,6].

This paper introduces an efficient thread-level parallel implementation for color space conversion on different multi-core processors. For this purpose, we allocate different input pixels to each thread by partitioning an image into multiple segments. The performance evaluation is made by comparing the processing speedup between its serial implementation and parallel ones.

The organization of this paper is as follows. Section 2 introduces the background of color conversion and thread-level parallelism. Section 3 discusses the implementation parallel color conversion. Section 4 covers the results of simulation about the implementation of parallel color conversion on different multi-core processors. In section 5, we conclude our result and discuss further perspectives.

Ⅱ. Background

Digital color images are represented as a set of three components representing the intensities of each of the three primaries: red, green, and blue (RGB). The color space obtained through combining the three colors can be determined by drawing a triangle on a special color chart with each of the base colors as an endpoint. Using the CIE chart as a guideline, NTSC defines the transmission of color signals in a luminance and chrominance format called YIQ. The YUV format, a variant of the YIQ format, concentrates most of the image information into the luminance (Y) and less in the chrominance (UV), allowing the chrominance to be specified less frequently than the luminance [7].

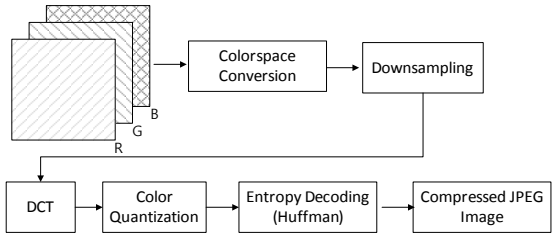

Fig. 1. Color conversion on JPEG

Therefore, in most color image and video compression algorithms, only every other U and V elements in the horizontal direction are sampled (4:2:2 format). The missing elements are reconstructed by either interpolation or duplication.

Figure 1 shows the basic JPEG compression method; it can be summarized as following: (1) the image is separated into three color components; (2) each component is partitioned into 8-by-8 blocks; (3) each block is transformed using the two dimensional DCT (Discrete Cosine Transform); (4) each transformed block is quantized with respect to an 8-by-8 quantization matrix; (5) the resulting data is compressed, using Huffman or arithmetic coding.

In order to achieve good compression performance, correlation between the color components is first reduced by converting the RGB color space into a de-correlated color space. In baseline JPEG, a RGB image is first transformed into a luminance-chrominance color space such as luminance-chrominance (L-C) color space such as YCbCr, YUV, CIELAB, etc [8].

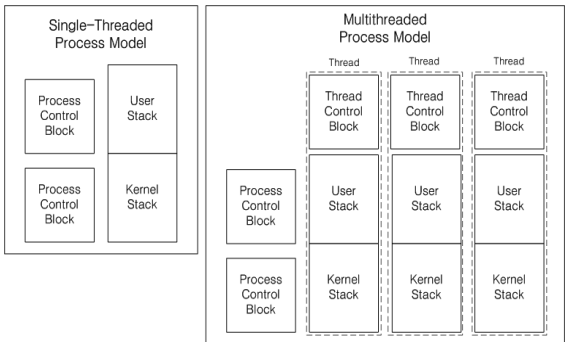

Fig. 2. Thread execution model

A thread is a lightweight process having its own program counter and execution stack as shown in Fig. 2. It shows the comparison between single-threaded and multi-threaded processing model. The model is very flexible, but low level, and is usually associated with shared memory and operating systems [4].

Ⅲ. Parallel Implementation

The actual conversion between the standard RGB format to the YUV format and vice-versa is slightly different for digital signals than for analog signals. The digital conversion requires that if the RGB values are between 0 and 255, the YUV values should also be between 0 and 255. Fig. 3 shows the block diagram of color conversion between RGB and YUV.

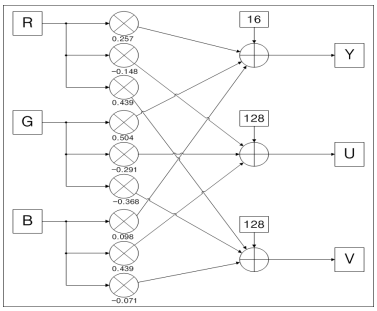

Fig. 3. Conversion between RGB and YUV

This method can be achieved by performing a matrix transformation of R, G, and B pixel data of a set of possible R, G, and B parameters into corresponding Y, U, and V parameters. This process is based on the bellowing mathematical expression:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.148 & -0.289 & 0.439 \\ 0.615 & -0.515 & -0.1 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (1)$$

The value Y = 0.299R + 0.587G + 0.114B is called the luminance. The formula is like a weighted-filter with different weights for each spectral component. Accordingly, the inverse transformation from YUV to RGB can be:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.0 & 0.0 & 1.13983 \\ 1.0 & -0.39465 & -0.58060 \\ 1.0 & 2.03211 & 0.0 \end{bmatrix} \begin{bmatrix} Y \\ U \\ V \end{bmatrix} \quad (2)$$

```
//Windows Thread
DWORD WINAPI threadwork(LPVOID arg)
{
        struct threadposition *tp = (struct threadposition *)arg;
        for(int i = tp->starty; i < tp->endy; i++){
                for(int j = tp->startx; j < tp->endx; j++){
                        double y = int2y(i_r[i][j], i_g[i][j], i_b[i][j]);
                        double u = int2u(i_b[i][j], y);
                        double v = int2v(i_r[i][j], y);

                        d_y[i][j] = y;
                        d_u[i][j] = u;
                        d_v[i][j] = v;
                }
        }
        threadStatus[tp->threadnum] = true
        return 0;
}
```

Fig. 4. Thread code block

Thread-level parallel implementation for the above equations is achieved by allocating different input pixels to each thread using Window Thread. Figure 4 shows the code block related with thread functions. The CreateThread function creates a new thread for a process shown in Figure 5. The creating thread must specify the starting address of the code that the new thread is to execute. Typically, the starting address is the name of a function defined in the program code This function takes a single parameter and returns a DWORD value. A process can have multiple threads simultaneously executing the same function.

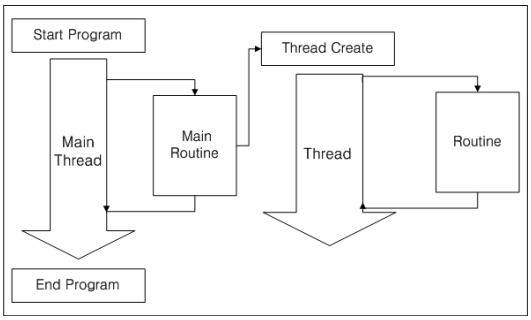

Fig. 5. Thread creations

Ⅳ. Simulation Results

This section describes the implementation environments and results in detail. For the performance evaluation of thread-level parallelled color space conversion over its serial one, the processing time was measured using three different thread numbers such as one, four, and eight threads. They are denoted as Thread 1, Thread 4, and Thread 8, respectively. Table 1 describes the specification of four multi-core systems.

As for the number of thread, we used 4 and 8 threads. Fig. 6 shows the processing times on four multi-core processors based on the number of threads. The results show 2 times of performance improvements on the processing speed compared with the serial implementation. Fig. 7 shows the improvement ratio with increasing the number of threads. The result shows that when thread number is same as the number of cores, the ratio becomes high.

Table 1. Simulation environment

Model	Specification
i3	4 core, 16GB RAM SSD
i5	4 core, 4GB RAM HDD
i7	4 core, 4GB RAM HDD
Phenom 960T	4 core, 4GB RAM HDD

Ⅴ. Conclusion

In general, color space conversion has become a very important role in the image acquisition, display and the

transmission of the color information. With the poularity of multi-core processors, thread-level parallelism has been becoming very useful parallel processing paradigm especially on shared memory computing systems. This paper experiments the performance of JPEG color space conversion of RGB to YUV using thread-level parallel implementation on different multi-core processors. The implementation results show 2 times of performance improvements on the processing speed compared with the serial implementation and all multi-cores show similar performance improvement ratios.

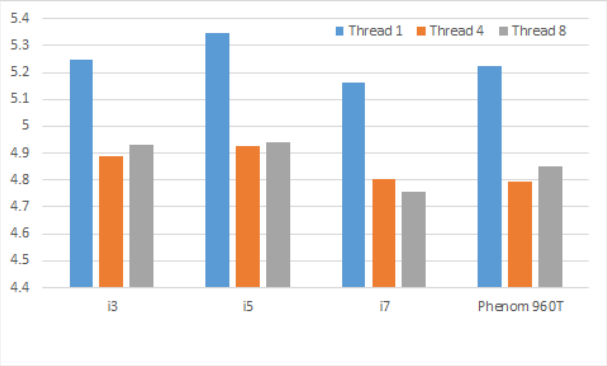

Fig. 6. Processing speed

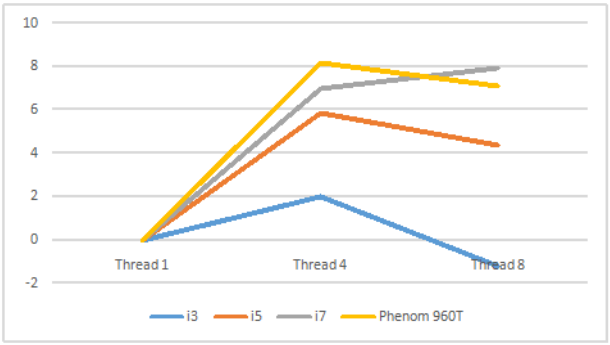

Fig. 7. Performance improvements ratio

References

[1] Cheong Ghil Kim, "Parallel SAD for Fast Dense Disparity Map Using a Shared Memory Programming", Information Technology Convergence, Vol. 2, pp. 1055-1060, Jul. 2013.

[2] Cheong Ghil Kim, Do Hyun Lee, JeomGu Kim, "Optimizing Image Processing on Multi-core CPUs with Intel Parallel Programming Technologies," Multimedia Tools and Applications January 2014, Vol. 68, Issue 2, pp 237-251, Jan. 2014.

[3] E. Ajkunic, H. Fatkic, E. Omerovic, K. Talic, and N. Nosovic, "A Comparison of Five Parallel Programming Models for C++", in Proc. of 35th Int'l Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO 2012), May 2012, pp. 1780 - 1784, 2012.

[4] Javier Diaz, Camelia Mun˜oz-Caro, and Alfonso Nino, "A Survey of Parallel Programming Models and Tools in the Multi and Many-Core Era", IEEE Transactions on Parallel and Distributed Systems, VOL. 23, NO. 8, pp. 1369-1386, Aug. 2012.

[5] Simone Campanoni, Timothy M. Jones, Glenn Holloway, Gu-Yeon Wei, and David Brooks, "Helix: Making the Extraction of Thread-Level Parallelism Mainstream", IEEE Micro, Volume: 32 , Issue: 4, pp. 8 - 18, 2012

[6] Dongrui Fan, Hao Zhang, Da Wang, Xiaochun Ye, Fenglong Song, Guojie Li, and Ninghui Sun, "Godson-T: An Efficient Many-Core Processor Exploring Thread-Level Parallelism", IEEE Micro, Volume: 32 , Issue: 4, pp. 38 - 47, 2012.

[7] Benjamin Gordon, Navin Chaddha and Teresa Meng, "A Low-Power Multiplierless YUV to RGB Converter Based on Human Vision Perception," Workshop on VLSI Signal Processing, VII, pp. 408 - 417, Oct 1994.

[8] T. Acharya and P. Tsai, "JPEG: Still Image Compression Standard," in JPEG2000 standard for image compression : concepts, algorithms and VLSl architecturcs, John Wiley & Sons, Inc., Hoboken, New Jersey, 2005.

저자

김 정 길(Cheong Ghil Kim) **종신회원**

- 1987년 8월 : Univ. of Redlands, USA 컴퓨터과학과 학사졸업
- 2003년 8월 : 연세대학교 컴퓨터과학과 공학석사 졸업
- 2006년 8월 : 연세대학교 컴퓨터과학과공학박사 졸업
- 2006년 ~ 2007년 : 연세대학교 컴퓨터과학과 박사후 연구원
- 2007년 ~ 2008년 : 연세대학교 컴퓨터과학과 연구교수
- 2008년 ~ 현재 : 남서울대학교 컴퓨터학과교수

<관심분야> : 멀티미디어 임베디드 시스템, 이기종 컴퓨팅, 모바일 AR, 3D Contents