고속 컬러 좌표계 변환을 위한 병렬 프로그래밍

최상근, 손채봉

광운대학교 전자통신공학과

{ hnes6522 | cbsohn }@kw.ac.kr

Parallel programming for high-speed color space conversion

Sang-Geun Choi and Chae-Bong Sohn

Department of Electronics and Communications Engineering, Kwangwoon University

요 약

YUV 파일을 RGB 형태의 color space 로 변환하는 과정은 엄청난 연산으로 많은 시간이 소요된다. 이런 문제를 다양한 방법을 이용하여 속도 감소율을 확인할 것이다. 처음으로 기본 소스코드의 소요시간을 기준으로 삼기 위하여 최적화와 병렬프로그래밍을 사용하지 않고 프로그램을 설계하였다.

최적화와 병렬프로그래밍 단계를 진행하였을 때 C언어로 구현 된 최적화되기 전과 최종적으로 CUDA 기반의 병렬프로그래밍을 사용한 함수를 비교해보았을 때 속도의 증가율이 575%로 엄청난 속도의 차이를 확인할 수 있다.

이와 같은 기술을 영상을 다루는 모든 분야에서 처리속도가 증가함에 따라 효과적인 작업을 기대해 볼 수 있다.

Ⅰ. 서론

영상 신호처리란 영상에 대한 디지털 정보를 신호로 인식하여 처리하는 방법이다. 디지털시대에 들어서면서 모든 정보는 디지털 정보로 표현이 가능하게 되었고, 그에 따라 디지털 신호를 처리하는 기술이 발전해 가고 있다. 신호를 처리하는데 있어서 속도는 가장 중요한 쟁점이다. 영상의 경우에는 기술의 발전에 따라 데이터의 양이 방대해지고 있기 때문에 보통의 신호처리에서 속도를 증가시키는 방법으로는 아직 턱없이 부족하다. 이런 문제를 해결하기 위해서 병렬 프로그래밍을 사용하는 것이다[1].

병렬프로그래밍이란 한 번의 명령으로 하나의 작업을 수행하는 것이 아닌 한 번의 명령에 여러 작업을 동시에 수행 가능하도록 하는 방법이다. 병렬프로그래밍은 CPU 또는 GPU의 코어를 이용하여 multi-threading 방법을 통하여 속도를 향상시킨다[2][3].

병렬 프로그래밍을 영상 신호처리에 이용하면 방대한 양의 데이터를 빠른 속도로 처리가 가능해져 영상 신호처리와 병렬 프로그래밍은 필수불가결한 관계이다.

Ⅱ. YUV 4:2:0, RGB 변환

1. YUV 4:2:0 파일 형식

YUV는 luminance와 chrominance로 구성되어 있는 color space 다. Y는 luminance를 UV는 chrominance를 나타내며 YUV 4:2:0의 경우 UV의 정보를 각각 4분의 1로 줄여 저장하는 방식이다. 이것이 가능한 이유는 인간의 눈은 색에 대한 민감도보다 밝기에 대한 민감도가 더 큰 영향을 미치기 때문이다. 밝기에 대한 정보는 그대로 전달하는 반면 색에 대한 정보는 4개의 픽셀의 평균값만을 보내서 원본 영상과 많은 차이를 주지 않으면서 데이터의 사이즈를 줄여줄 수 있다[4][5].

Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
U	U	U	U	U	U	U	U	U	U
V	V	V	V	V	V	V	V	V	V

그림 1. YUV 4:2:0 의 일반적인 파일 형식
Fig. 1. General YUV 4:2:0 file format

YUV 4:2:0 파일은 위의 그림처럼 Y에 대한 정보를 먼저 나열한 뒤 U와 V의 정보를 차례대로 쌓여있는 구조를 가지고 있다.

2. YUV 4:2:0, RGB 상호 변환

YUV 4:2:0 file은 RGB의 값을 변환하여 저장하는 방식이기 때문에 역으로 RGB의 값으로 반환을 해줄 수 있다.

식 (1)은 RGB좌표계에서 YUV좌표계로 변환하기 위한 수식이다. 식 (2)는 YUV좌표계에서 RGB좌표계로 변환하기 위한 수식이다[4][5].

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.257 & 0.504 & 0.098 \\ -0.148 & -0.291 & 0.439 \\ 0.439 & -0.368 & -0.071 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} \quad (1)$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.164 & 0 & 1.596 \\ 1.164 & -0.391 & -0.813 \\ 1.164 & 2.018 & 0 \end{bmatrix} \left(\begin{bmatrix} Y \\ U \\ V \end{bmatrix} - \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} \right) \quad (2)$$

Ⅲ. 프로그램 기본 흐름도

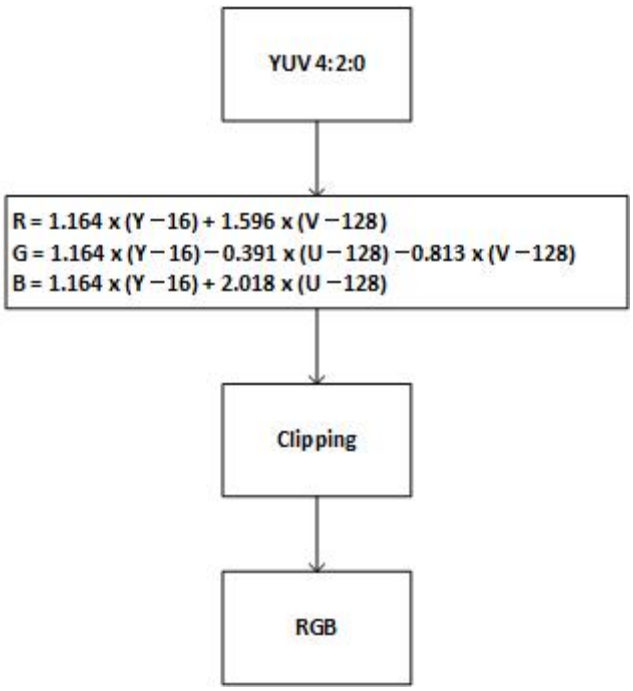

그림 2. YUV 4:2:0, RGB 변환 흐름도
Fig. 2. YUV4:2:0 to RGB conversion flowchart

그림 2와 같이 YUV에서 RGB로 변환에 필요한 수식을 보면 많은 소수점 연산과 곱셈 연산을 보이고 있다. 연산을 수행할 때 소수점 연산과 곱셈 연산은 정수 연산과 덧셈 연산에 비하여 많은 시간이 소요된다. 이러한 문제로 데이터의 크기가 커짐에 따라 프로그램의 동작 시간은 점점 늘어나게 된다[5].

또 클리핑을 해주는 단계에서 조건에 따라 값을 재설정 해주어야 한다. 이 과정에서 조건문을 사용하게 되는데 이는 다음에 수행될 작업이 조건에 따라 달라져 파이프라인에 쌓여있던 것들을 깨트려 속도 저하의 원인이 된다. 고속으로 처리하는 부분에서 파이프라인은 절대 깨져서는 안되는 중요한 역할을 한다. 이런 문제를 해결하기 위한 방법이 최적화와 병렬프로그래밍이다[5].

본 논문에서는 최적화에 대한 내용은 다루지 않고 SIMD기반의 병렬프로그램과 CUDA기반의 병렬프로그램을 사용하여 멀티 스레딩 방식을 구현하였다.

Ⅳ. SIMD 병렬프로그램 적용

SIMD 프로그램은 CPU의 코어를 활용하여 병렬프로그래밍을 가능하게 해주는 프로그램의 하나이다. SIMD의 장점으로는 CPU의 코어를 직접 활용하기 때문에 빠른 속도로 메모리를 교환할 수 있다는 장점을 가지고 있다. 반면 CPU의 코어는 GPU의 코어에 비해 개수가 적다는 단점을 가지고 있어 동시에 많은 양의 데이터를 처리할 수가 없다. 그래서 SIMD프로그램은 방대한 데이터를 처리하기보다 일반적인 데이터의 크기를 처리할 경우에 적합하다[5][6].

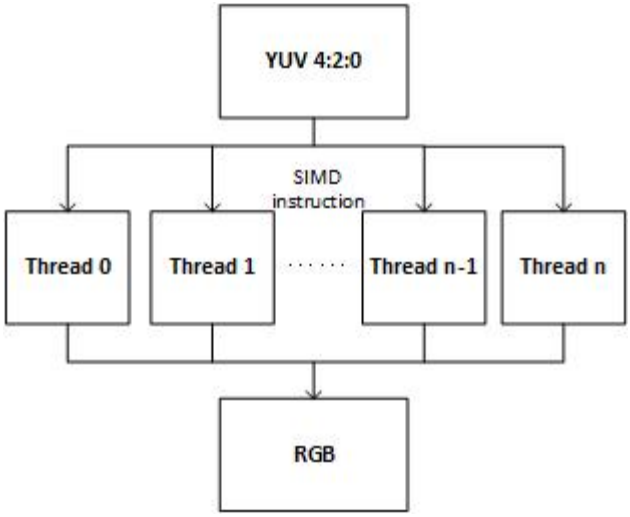

그림 3. SIMD 프로그램을 적용한 흐름도
Fig. 3. Flowchart applying the SIMD program

그림 3과 같이 여러 개의 스레드가 동시에 동작을 하여 연산을 수행하게 된다. 여기서 특이한 점은 SIMD 프로그램은 어셈블리어를 사용하여 코드를 작성한다. 어셈블리어는 기계어에 가까운 언어이다. 어셈블리어를 사용함으로써 컴퓨터가 보다 빠른 반응을 할 수 있게 해주는 장점이 있다. 반면 기계어에 가깝다보니 가독성이 떨어져 생산성면에서 떨어진다는 단점이 있다[5][6].

그림 2의 흐름도와 차이가 있는 부분이 클리핑 부분이다. SIMD에서는 기본적으로 제공해주는 함수 중에 연산과 동시에 클리핑을 함께 수행해주는 함수가 있어 별도의 클리핑작업을 수행하지 않아도 되기 때문에 속도증가에 도움을 주고 있다[5].

Ⅴ. CUDA 병렬프로그램 적용

CUDA프로그래밍은 NVIDIA에서 제공하는 병렬프로그래밍으로 GPU를 사용하는 것이 큰 특징이다. GPU는 CPU와는 비교할 수 없을 정도로 많은 코어를 가지고 있어 한 번에 엄청난 양의 데이터를 동시에 처리가 가능하다. 반면 GPU를 사용하기 때문에 메모리의 이동에 걸리는 시간이 길다는 단점이 있다. CUDA프로그래밍에서 중요하다고 생각되는 것은 스레드를 얼마나 효율적으로 사용하고 효율적으로 배치하는가에 있다.[2][3]

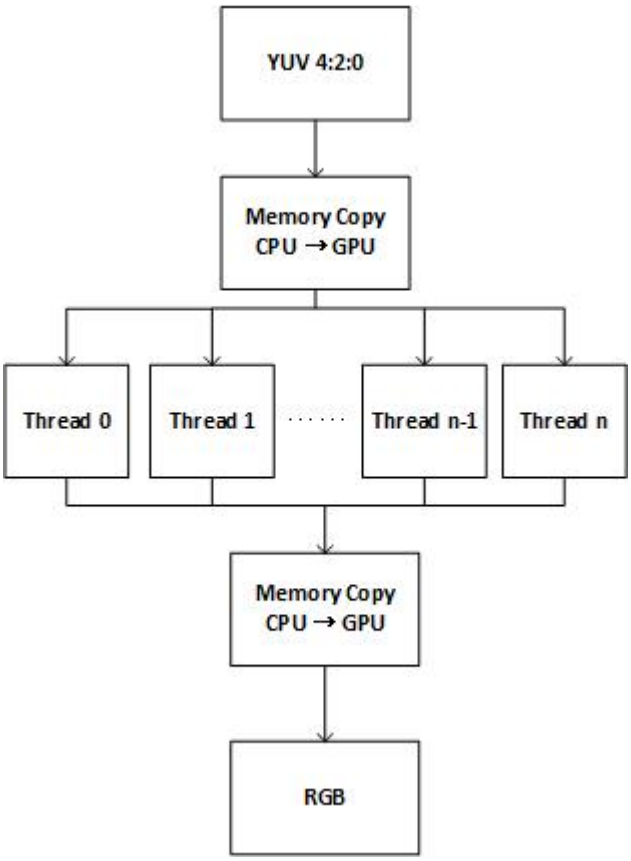

그림 4. CUDA 프로그램을 적용한 흐름도
Fig. 4. Flowchart applying the CUDA program

그림 4와 같이 CPU에서 GPU로 YUV의 데이터를 넘겨주고 있다. 이 과정에서 많은 시간이 소요되기 때문에 적은 양의 데이터를 CUDA 프로그램을 적용시켜 동작을 할 경우에 데이터 이동에 소요되는 시간이 멀티 스레딩을 통해 얻는 시간이득보다 커지게 돼서 결과적으로 이득을 얻지 못하게 된다[2][3].

데이터의 전송이 끝난 뒤에 여러개의 스레드가 동시다발적으로 연산을 시작하게 된다. 하나의 스레드로 전체 연산을 진행하는 방식은 많은 시간이 소요되지만 여러개의 스레드를 사용하여 연산을 진행하게 되면 스레드 개수에 따라 연산에 소요되는 시간이 줄어들게 된다.

마지막으로 연산이 종료되면 RGB데이터를 다시 CPU로 넘겨주어 프로그램을 종료하게 된다.

Ⅵ. 시뮬레이션

표 1. 함수 수행 소요시간 및 fps
Table 1. Function run-time and fps

	1	2	3	4	5	avr	fps
Basic	38.11	38.06	38.20	38.47	38.23	38.2	6.2
SIMD	14.79	14.82	14.95	14.79	14.92	15.0	15.9
CUDA	5.649	5.641	5.625	5.625	5.812	5.6	42.4

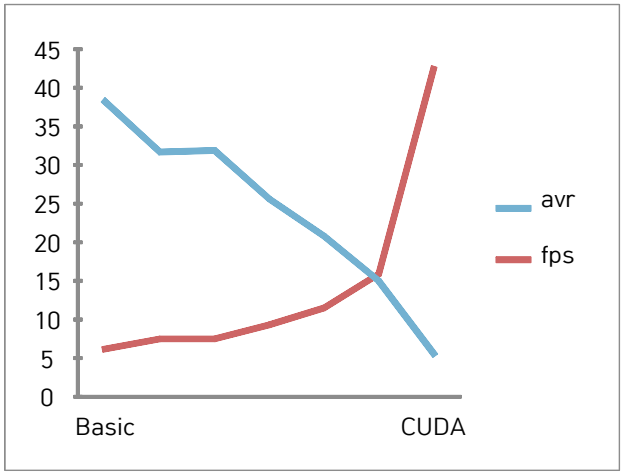

그림 5. 최적화에 따른 속도 및 fps 변화
Fig. 5. Speed and fps change according to the optimization

테스트에 사용한 영상의 사이즈는 1920x1080의 hd사이즈의 YUV 파일을 사용하였다. 이 영상은 초당 재생되는 프레임의 수는 24fps의 수준을 요구한다. 프로그램 동작 환경은 CPU는 코어 i5-2410M(2.3GHz 듀얼코어)를 GPU는 GeForce GT 540M을 사용하였다. 이번 테스트에서는 240프레임씩 돌려보면서 샘플을 추출해 보았다.

단일 스레드로 연산을 하였을 경우 240프레임을 수행하는데에 평균 38초의 시간이 소요되었고 이것을 fps로 환산을 하였을 때 6.3fps의 수준을 갖는다. 수치상으로만 봐도 턱없이 부족한 수준이다.

SIMD를 사용한 함수는 240프레임에 15초의 시간이 소요되었고 15.9fps의 수준을 보이고 있다. CPU의 코어를 사용하여 병렬프로그램을 수행하였을 때 단일스레드에 비하여 2배 이상의 속도 향상을 보이고 있다.

CUDA를 사용한 함수는 240프레임에 5.66초의 시간이 소요되었고 42.4fps의 수준을 보이고 있다. GPU의 코어를 사용하여 병렬프로그램을 수행하였을 때 CPU와 GPU간의 데이터 전송에 많은 시간일 걸림에도 불구하고 단일스레드와 비교하였을 때 5배 이상의 속도향상을 SIMD프로그램과 비교하였을 때 약 3배의 속도향상을 보이고 있다. 이는 코어의 개수가 방대한 GPU를 사용함으로 엄청난 수의 스레드를 이용하여 동시에 많은 양의 데이터를 처리하기 때문에 가능한 결과이다.

Ⅷ. 결론

시뮬레이션 결과를 봤을 때 싱글 스레딩을 통한 함수의 처리속도와 멀티 스레딩을 통한 함수의 처리속도의 차이가 SIMD의 경우 156%의 속도이득을 보이고, CUDA의 경우 583%를 보이고 있다.

따라서 본 논문에서 제안된 SIMD, CUDA기반의 병렬프로그램을 통한 방대한 데이터의 처리 방법은 점차 CPU와 GPU 성능의 향상과 병렬처리에 관한 알고리즘이 개선됨에 따라 방대한 데이터 처리 기술 발전에 크게 기여할 수 있는 방법으로 기대된다.

참고문헌

[1] J.B. Lee, “Performance Study of Multicore Digital Signal Processor Architectures”, 2013

[2] Morgan Kaufmann, Shane Cook, CUDA Programming : A developer’s guide to parallel computing with GPUs, 2012

[3] Addison-Wesley, Sanders, Jason, CUDA By Example : an introduction to general-purpose GPU programming, 2010

[4] Zaher Hamid Al-Tain, Rahmita Wirza Rahmat, M. Iqbal Saripan, Puteri Suhaiza Sulaiman, “Skin Segmentation Using YUV and RGB Color Spaces”, 2014

[5] Etienne Dupuis, “Optimizing YUV-RGB Color Space Conversion Using Intel’s SIMD Technology”, August 2003

[6] H.K. Yun, C.M. Koo, D.T. Moon, “Design of High Speed Floating-Point Arithmetic Module for SIMD Parallel Processor”, 2012