한국마린엔지니어링학회지 제34권 제6호, pp. 858~863, 2010. 9 (ISSN 1226-9549)
Journal of the Korean Society of Marine Engineering

CUDA 기반의 병렬 프로그래밍을 통한 H.264/AVC 부호화 속도 향상 및 CPU 부하 경감

장은빈[1] · 하윤수†

(원고접수일 : 2010년 6월 11일, 원고수정일 : 2010년 7월 26일, 심사완료일 : 2010년 8월 6일)

Enhancement of H.264/AVC Encoding Speed and Reduction of CPU Load through Parallel Programming Based on CUDA

Eun-Been Jang[1] · Yun-Su Ha†

요 약 : H.264/AVC를 이용한 동영상의 부호화에서 그 속도를 높이기 위해서는 움직임 예측시간을 줄이는 것이 매우 중요하다. 본 논문에서는 H.264/AVC 부호기의 오픈 소스인 x.264를 대상으로 움직임 예측 알고리즘을 CUDA 기반에서 구현함으로서 기존의 압축 기술 이상의 속도 향상 및 CPU의 점유율을 경감 시킬 수 있음을 검증한다.
주제어 : 부호화 속도, 움직임 예측, CUDA, CPU 점유율

Abstract: In order to enhance encoding speed in dynamic image encoding using H.264/AVC, reducing the time for motion estimation which takes a large portion of the processing time is very important. An approach using graphics processing unit(GPU) as a coprocessor to assist the central processing unit(CPU) in computing massive data, will be a way to reduce the processing time. In this paper, we present an efficient block-level parallel algorithm for the motion estimation(ME) on a computer unified device architecture(CUDA) platform developed in general-purpose computation on GPU. Experiments are carried out to verify the effectiveness of the proposed algorithm.
Key words: Encoding Speed, Motion Estimation(ME), Graphics Processing Unit(GPU), General Purpose computing on Graphics Processing Units(GPGPU), Computer Unified Device Architecture(CUDA)

1. 서 론

H.264/AVC는 높은 압축 성능과 뛰어난 화질을 보이는 현존하는 가장 우수한 비디오 부호화 표준[1-2]이다. H.264/AVC는 동화상 부화화 방식에서는 종래의 동화상 부호화 방식과 마찬가지로 이미 부호화된 화상 프레임으로부터의 움직임을 추정하여 예측 신호를 작성하고, 잔차 신호를 이산 코사인 변환(DCT)하여 부호화하는 기술을 토대로 하고 있으나, 움직임 예측알고리즘의 개량 등을 통해 종래보다 약 2배의 압축 성능을 실현하고 있다. 한편, 높은 압축 효율과 좋은 영상 품질을 가지기 위해서는 움직임 예측 시, 필요한 움직임을 찾는 연산 양이 많아지고 연산 시간이 늘어나는 문제점을 안고 있다. 따라서 동영상의 부호화 효율 및 성능을 보다 높이기 위해서는 움직임 예측에 필요한 시간을 줄이는 연구가 필요하다. 최근 병렬 컴퓨팅을 C언어 기반에서 가능하게 하는 CUDA (Compute Unified Device Architecture)라는

† 교신저자(한국해양대학교 IT 공학부, E-mail:hys@hhu.ac.kr, Tel: 010-4579-4347)
1 한국해양대학교 IT공학부

라이브러리를 이용하여 분기문 또는 과거 연산에 영향을 받는 종속적인 부분이 없는 대량의 연산에 대해 CPU가 처리하는 연산들을 GPU (Graphics Processing Unit)가 대신 처리하도록 함으로써 처리속도를 향상시키고자하는 연구가 시도되고 있다[3-4]. 이러한 노력의 하나로, 본 논문에서는 H.264/AVC 부호기의 오픈 소스인 x.264를 대상으로 움직임 예측(Motion Estimation)알고리즘을 CUDA 기반에서 구현하여 기존의 압축 기술 이상의 속도 향상 및 CPU 점유율을 경감 시킬 수 있음을 검증한다.

검증실험은 보급형 그래픽카드인 Geforce 8600GT와 고성능 그래픽카드인 GTX260 환경에서 해상도가 각각 QCIF (176 × 144), CIF(352 × 288), FULL HD(1920 × 1080)인 동영상을 대상으로 이루어진다.

2. 시스템의 구성

2.1 전체 시스템 구성도

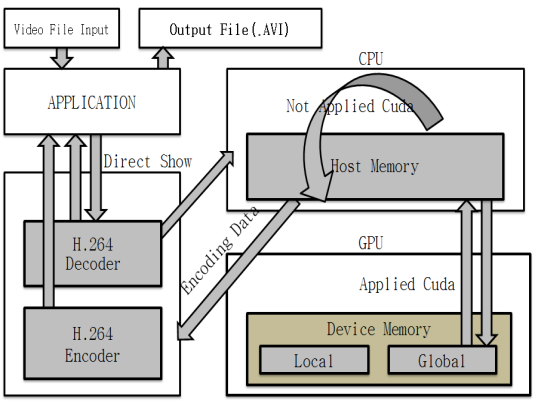

Figure 1: System architecture

Figure 1은 본 연구에서 대상으로 하고 있는 CUDA를 적용한 H.264/AVC 부호기의 전체 시스템 구성도이다. 우선 HD급 화질의 파일이나 기타 동영상 파일을 입력으로 받아 응용 프로그램에서 부호화 작업이 시작되면 연산들을 CPU가 단독으로 처리하거나 병렬 처리가 가능한 연산에 대해서는 CUDA를 적용한 코드가 구현되어 있는 Direct Show Filter를 통해 데이터를 CPU에서 GPU로 이동시킨다. 이동된 데이터는 GPU에서 CUDA 기반의 연산을 거쳐 처리되며, 그 결과를 CPU로 넘겨받는 과정을 통해 부호화한다. CUDA 적용에 대한 유효성 검증을 위하여, 부호기는 CUDA를 적용한 것과 사적하지 않은 것으로 나뉘어진다.

2.2 H.264 부호기 구성도

Figure 2는 H.264/AVC 부호기의 구조와 본 연구에서 CUDA를 적용할 부분에 대해 나타낸 구성도이다. 필터는 Direct Show를 기반으로 제작하고, 내부에 있는 코드는 CUDA를 기반으로 하여 C 프로그래밍을 하게 된다.

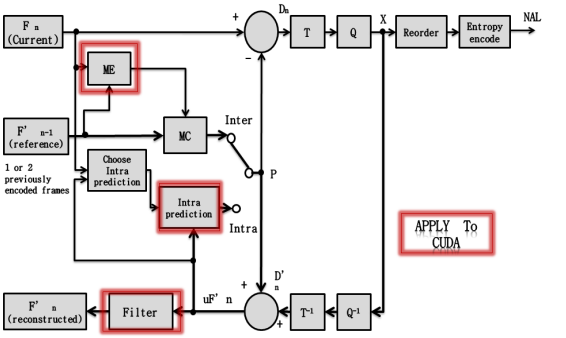

Figure 2: H.264 encoder architecture

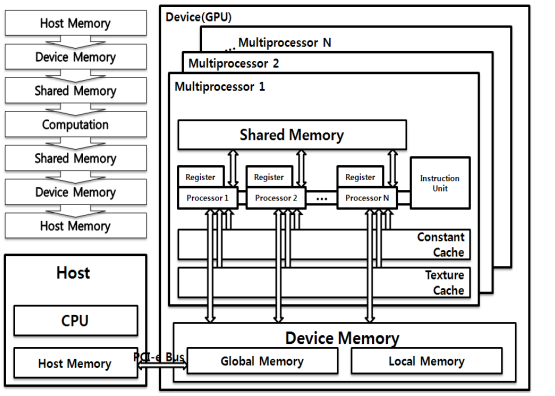

Figure 3: CUDA hardware architecture

2.3 CUDA 하드웨어 구성도

하나의 GPU 안에는 여러 개의 스트림 멀티프로세서(SM)이 있고 각 SM에는 여러 개의 스트림

프로세서(SP)가 있다. SP는 Figure 3과 같이 실행되며 다양한 형태의 메모리들을 가지게 된다.

- 프로세서마다 32비트 레지스터들의 세트 1개.
- 멀티프로세서 안의 모든 프로세서 코어들이 공유하는 공유 메모리.
- 모든 프로세서 코어들에 공유되고 상수 / 텍스쳐 메모리 공간의 읽기속도를 향상 시켜주는, 읽기 전용인 상수 / 텍스쳐 캐시.

CUDA 기반의 프로그래밍에서 데이터의 이동은 CPU 메모리 → GPU 메모리 → 공유 메모리 → 연산 → 공유 메모리 → GPU 메모리 → CPU 메모리의 순으로 실행된다.

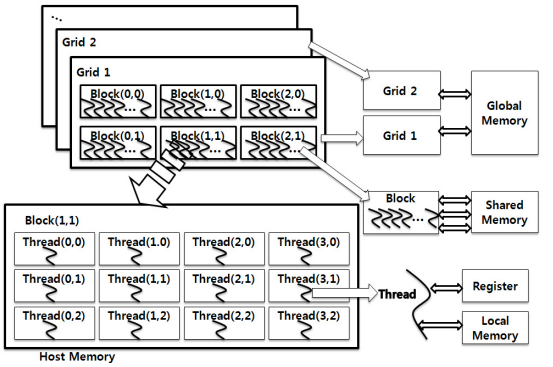

Figure 4: GPU stream processor architecture

Figure 4에서 스레드라고 되어 있는 것들이 CUDA 기반의 연산에서 가장 작은 단위인데 이 스레드들은 블록의 하위에 있고 블록들은 다시 그리드의 하위에 있다. 그래서 프로그래머가 연산을 하려면 '몇 번 그리드의 몇 번 블록의 몇 번 스레드를 사용하여 계산 하겠다'라고 지정해 주어야 한다. 여기서 각 스레드는 자신만의 지역 메모리와 레지스터를 가지게 되며 읽기, 쓰기가 가능하다. 각 스레드 간에 데이터 공유를 위한 공유 메모리는 블록 별로 존재하여 여러 스레드가 공유하며 읽기, 쓰기가 가능하다. 그리고 블록들의 집합인 그리드 간에 데이터 공유를 위한 전역 메모리는 읽기, 쓰기는 되지만 캐시가 되지 않는다. 상수 메모리와 텍스쳐 메모리 역시 그리드 별로 존재하지만 상수 메모리는 캐시가 되고 텍스쳐 메모리는 일부만 캐시가 된다.

이 메모리들을 어떻게 사용하느냐에 따라서 같은 연산을 하는 CUDA 응용 프로그램이라 하더라도 큰 성능차이를 내게 된다. 메모리간의 데이터를 이동시키는 비용이 꽤 크기 때문에 적절한 알고리즘을 작성하지 않는다면 계산하는데서 단축한 시간을 데이터를 이동시키는 데서 다 소비할 수도 있다. 각 메모리들의 속도 또한 차이가 있어서 이런 여러 가지 요소를 고려하여야 한다. 결국 CUDA 프로그래밍의 핵심은 각 메모리들의 특성을 파악해서 최대한 효율적인 알고리즘을 만드는데 있다.

3. 움직임 예측 알고리즘에 대한 CUDA의 적용

움직임예측(ME:Motion Estimation) 알고리즘이란 매크로 블록(16×16 Pixel) 단위의 움직임 추정을 말하는데, 현재 프레임과 인접한 프레임은 비슷한 형태로 이루어져 있기 때문에 모든 데이터를 가지고 있을 필요 없이 현재 프레임은 이전 프레임의 어느 부분과 비슷하다는 식의 표시만 해주는 방법으로 데이터의 양을 줄일 수 있다.

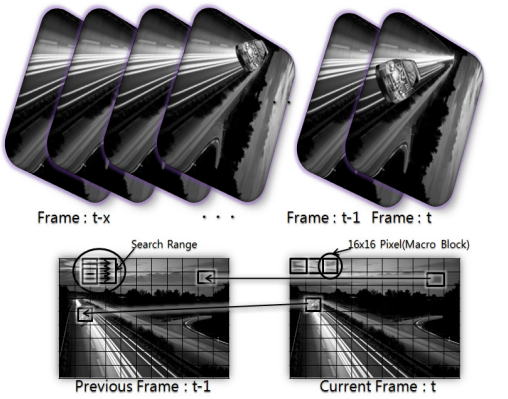

Figure 5: Motion estimation algorithm

Figure 5에서와 같은 동영상이 있다면 전체적인 도로 배경은 거의 바뀌지 않기 때문에 자동차의 이동에 따른 프레임 변화만 나타낸다면 동영상의 용량을 효율적으로 줄일 수 있을 것이다. ME는 보통 매크로 블록 단위에 대해 현재 프레임이 이전 프레임의 같은 좌표의 매크로 블록에서 크게 벗어나지 않을 것이므로 검색 범위를 매크로블록 크기의

+16 정도로 주었을 때 검색 범위 내에서 최고의 매칭 포인트를 검색하게 된다. SAD(Sum of Absolute Difference)는 현재 프레임의 매크로 블록을 결정짓기 위해서 이전 프레임의 매크로 블록이 현재 프레임으로 바뀔 때 어느 위치로 이동했는지를 결정할 수 있는데, 이는 이전 프레임과 현재 매크로 블록에서 각 픽셀 간 차이 값에 절대 값을 취하여 구할 수 있다. 이는 만약에 프레임이 변경되지 않았다면 매크로 블록 간 차이 값을 뺀 값에 절대 값을 취해도 0이 되지만 조금이라도 움직였다면 검색하고 있는 매크로 블록마다 SAD값이 조금씩 틀려질 것이다. 결국 SAD를 통한 움직임 벡터 검출과 같이 SAD값이 최소가 되는 매크로 블록이 최고의 움직임 추정 매크로 블록이 된다.

H.264/AVC 압축 코덱의 오픈 소스인 x.264에 CUDA를 적용시키기 위해서 먼저 Figure 6과 같이 x.264의 성능을 분석하였다. 가장 많은 작업을 발생시키는 함수로는 x264_me_search_ref (struct x264_t*, struct x264_me_t*, short(*)[2], int, int*) 함수가 있는데 이 함수는 ME에서 각각의 매크로블록에 대한 참조프레임과의 모션을 분석하여 모션 벡터를 생성하는 함수이다. 함수 내에는 부화소에 대한 ME와 모드를 결정할 수 있는 알고리즘으로 SAD, 하다마드변환(SATD:Sum of Transformed Differences)등과 같은 방법이 있는데, 개별 함수로 SAD 모드 결정 방법이 가장 많이 실행되는 것으로 나타났으며,

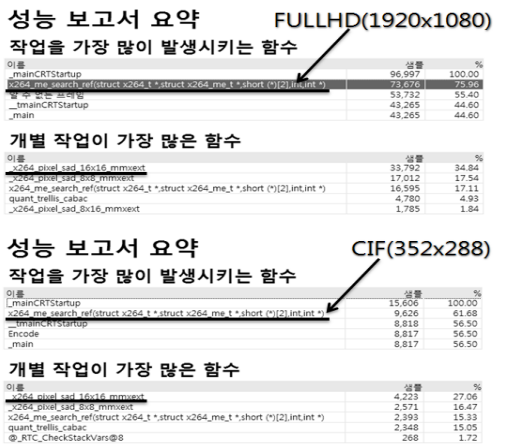

Figure 6: x.264 performance report

결과적으로 부호화 작업 중 ME 관련 연산이 60%이상의 부하를 주는 연산 작업인 것을 알 수 있다[5]. 이는 ME 알고리즘의 다양한 연구를 통해 압축 성능의 향상을 가져올 수 있음을 의미한다[6-7].

CIF(176 × 144) 영상의 경우 16 × 16 단위의 매크로 블록이 가로로 11개, 세로로 9개가 존재하는데 총 99개의 매크로 블록에 대해서 전역검색(Full Search)을 할 경우 하나의 매크로 블록이 256[가로 픽셀 범위(16) × 세로 픽셀 범위 (16)] 번의 SAD 연산을 수행하게 된다. 따라서 한 프레임에 대해 모션 예측을 할 경우 SAD 연산 횟수는 (176 / 16) × (144 / 16) × (16 × 16) × (16 × 16)번으로 총 6,488,064회의 연산을 수행한다. 만약 Full HD(1920 × 1080) 화질의 동영상을 부호화 하게 되면 SAD 연산 횟수는 (1920 / 16) × (1080 / 16) × (16 × 16) × 16 × (16 × 16) 로 534,773,760회가 된다. 만약 CPU가 동영상의 매 프레임마다 이 연산을 순차적으로 처리하게 될 경우 부하가 매우 증가하게 되며 부호화에 엄청난 시간이 소비된다. 따라서 본 논문에서는 SAD 연산은 한 프레임에 대해 독립적인 연산이 가능하다는 점을 고려하여 연산이 병렬 처리될 수 있도록 CUDA를 적용하였다.

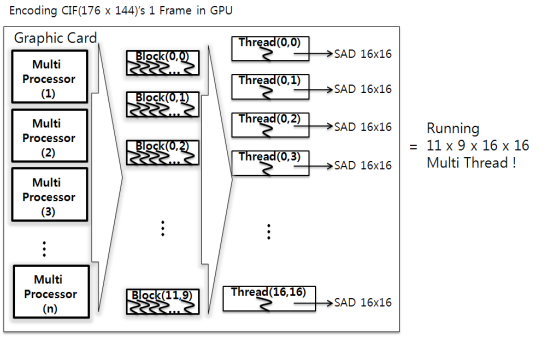

Figure 7: Turn SAD into CUDA

4. 실험 및 고찰

4.1 실험 환경

Figure 8은 본 논문에서 사용할 그래픽 카드의 사양을 Nvidia에서 제공하는 드라이버 API를 사용하여 출력한 것이다. 고성능 그래픽 카드인

GTX260의 경우 전역 메모리가 1GByte이며, 멀티프로세서의 개수는 27개, ALU는 216(하나의 멀티프로세서에 8개씩의 ALU가 존재)으로 독립적인 216개의 명령이 동시에 병렬 처리가 가능하다. 보급형 사양인 8600GT는 멀티프로세서가 4개뿐이며, ALU 또한 32개로 병렬 처리 속도 또한 GTX260에 비해 떨어짐을 알 수 있다.

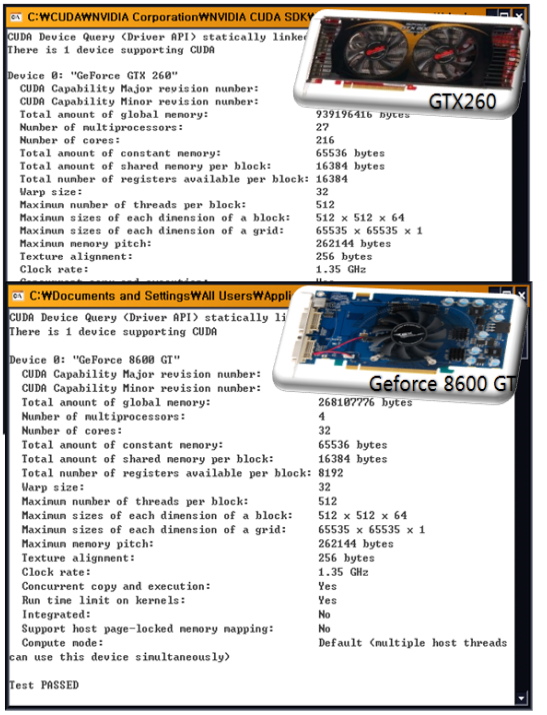

Figure 8: GPU specifications

부호화 테스트를 위한 CPU 사양은 다음과 같다.

- CPU : AMD Athlon(TM) 64 X2 Dual Core Processor 4000+ 2.11GHz
- RAM : 2.00GB

X.264 코덱의 경우 CPU의 멀티 스레드와 SIMD(Single Instruction Multiple Data) 명령어를 지원하기 때문에 CPU Core의 수와 성능에 따라 속도가 달라지므로 하나의 CPU를 대상으로 GPU의 사양에 따른 테스트가 불가피 하다. 따라서 CUDA 적용 유무에 따라 동일 CPU에 대해서 GPU를 달리하여 GPU 코어의 개수와 클럭 속도에 따른 부호화 속도의 변화를 측정하였다.

4.2 실험 결과 및 검토

Table1, Table2 는 서로 다른 GPU 환경에서 해상도가 각각 QCIF (176 × 144), CIF(352 × 288), FULL HD(1920 × 1080) 인 동영상을 대상으로 CUDA를 적용했을 때와, CPU명령어만을 사용했을 때, 그리고 멀티 스레드 기능을 추가로 적용하여 부호화한 결과를 비교한 표이다.

Table 1: Measured values of Geforce 8600GT

GPU : GEFORCE 8600 GT

PIXEL / STATUS	176 X 144	352 X 288	1920 X 1080	CPU's Occupation Rate
CUDA ON	112 fps	45 fps		50%
1Core/ MMX Off	1.5 fps	1.2 fps		50%
1Core/ MMX On	57 fps	37 fps		50%
Multithread/ MMX Off	3 fps	1.7 fps		100%
Multithread/ MMX On	130 fps	77 fps		100%

Table 2: Measured values of Geforce GTX260

GPU : GTX 260

PIXEL / STATUS	176 X 144	352 X 288	1920 X 1080	CPU's Occupation Rate
CUDA ON	100 fps	48 fps	2.17 fps	50%
1Core/ MMX Off	1.3 fps	1 fps		50%
1Core/ MMX On	57 fps	34 fps	2.10 fps	50%
Multithread/ MMX Off	2 fps	1.5 fps		100%
Multithread/ MMX On	98 fps	60 fps	2 fps	100%

Table 1과 같이 보급형 그래픽 카드인 Geforce 8600GT를 사용했을 때, 본 논문에서 제안한 CUDA를 적용한 부호화 결과는 속도 면에서 MMX명령어를 적용한 부호화 결과보다 현저히 개선된 것으로 나타났으나 멀티스레드를 적용했을 때보다는 다소 느린 것을 알 수 있다. 이는 그래픽 카드 사양이 병렬 처리를 수행하는 GPU내부 프로세서가 최대 32개 밖에 없기 때문에 연산 자체는 빠르지만 CPU에서 GPU로 메모리를 복사하는데 발생하는 시간 지연으로 인해 멀티스레드와 CPU 명령어를 사용해 처리하는 부호화 속도에 비해 속도가 다소 느린 결과를 보였다. 반면에 Table2에서와 같이 GTX260 환경에서는 CUDA를 적용한 부호화 결과가 멀티스레드와 CPU명령을 사용해 부호화한 결과보다 빠른 결과를 보임을 알 수 있다. 또한 해상도가 높을수록 매크로 블록의 개수가

많기 때문에 한 프레임에 대해 보다 빠르게 병렬 처리를 하기 때문에 Full HD 동영상 부호화 시 보다 빠른 결과를 나타내었다. 한편, CPU점유율에 있어서는 멀티스레드와 CPU명령을 사용한 경우 두 개의 대상 그래픽카드 환경에서 모두 100%였으나 CUDA를 적용하였을 때는 50%를 유지하는 결과를 보였다. 이는 CUDA를 적용했을 때, 연산 량이 많은 함수에 대해 GPU에서 연산을 병렬처리하고 CPU에서는 단순히 데이터를 GPU에 복사하고 가져오는 등의 작업 외에 다량의 연산을 하지 않기 때문에 CPU 명령과 다중 CPU Core를 사용한 멀티 스레드를 사용했을 때 보다 CPU가 가지는 작업 부하가 적다는 것을 나타낸다.

5. 결 론

본 논문에서는 GPU의 병렬 컴퓨팅 언어인 CUDA를 사용하여 H.264/AVC의 부호화 속도를 향상시키는 동시에 CPU의 부하를 경감시키는 방법을 제안 하였다.

제안된 방법에 대한 검증은 보급형 그래픽카드인 Geforce 8600GT와 고성능 그래픽카드인 GTX260 환경에서 해상도가 다른 3가지의 동영상을 대상으로 부호화 실험을 통하여 이루어졌으며, 그 결과 다음과 같은 결론을 얻었다.

첫째, 동일한 CPU 환경에서 부호화 작업 시 CPU가 처리해야 하는 연산의 대부분을 GPU에서 대신 처리하기 때문에 점유율 하락이 가능하다.

둘째, 동일한 CPU 환경에서 GPU의 사양이 보다 높을수록 프로세서의 양이 늘어나고, 이에 따라 동시에 처리할 수 있는 연산의 수가 많아짐으로 속도 향상이 가능하다.

셋째, 동영상 부호화 작업 시 프레임의 크기가 클수록 병렬 처리에 효율적이므로 영상의 화질이 높을수록 CUDA를 적용한 부호화 효율이 극대화 된다.

따라서 본 논문에서 제안된 CUDA 기반의 병렬 프로그래밍을 통한 동영상 압축 방법은 점차 GPU 성능의 향상과 병렬 처리 알고리즘이 개선됨에 따라 동영상 압축 표준의 발전에 크게 기여할 수 있는 구체적인 방안으로 기대된다.

참고문헌

[1] lain E.G Richardson, H.264 & MPEG-4 차세대 영상압축기술, 홍릉과학출판사, 2004.

[2] x264. http://en.wikipedia.org/wiki/X264.

[3] The GPGPU Resources and Forums, http://www.gpgpu.org/.

[4] NVidia Corp. CUDA Zone. http://www.nvidia.com.

[5] S.-Y Chien, Y.-W Huang, C.-Y Chen, H.-H Chen and L.-G Chen, "Hardware architecture design of video compression for multimedia communication systems," IEEE Commun. Mag.,vol. 43, no. 8, pp. 122-31, 2005.

[6] C.-W Ho, O. Au, S.-H. Chan, S.-K Yip and H.-M Wong, "Motion estimation for H.264/AVC using programmable graphics hardware", Proc. IEEE International Conference on Multimedia and Expo, pp. 2049-2052, 2006.

[7] R.-X. Chen and J. Fan, "Complexity reduction for SOPCbased H.264/AVC coder via sum of absolute difference", Proc. IEEE/CIE 7th International Conference on ASIC, pp. 1277-1280, 2007.

저 자 소 개

장은빈(張銀斌)

현재 한국해양대학교 IT공학부 제어자동화공학 전공 재학중. 관심분야: 임베디드 시스템, 로봇, 센서

하윤수(河潤秀)

1990년 한국해양대학교 대학원 졸업(공학석사). 1996년 일본 쓰쿠바대학 대학원 졸업 (공학박사). 1996~현재 한국해양대학교 IT공학부 교수