H.264의 복호화기를 위한 SIMD기반의 효율적인 고속 역 변환 방법

*유상준, **김성훈, **오승준, **손채봉, **안창범, **박호종
*, **광운대학교 VIA 멀티미디어 센터
*yusj1004@media.kw.ac.kr

An Efficient high-speed reverse conversion method of the SIMD base for the decoder of the H.264

*Sang-Jun Yu, **Seong-Hoon Kim, **Seoung-Jun Oh, **Chae-Bong Sohn,
**Chang-Beom Ahn, and **Ho-Chong Park
*,**VIA-Multimedia Center, Kwangwoon university

본 논문에서는 SIMD 명령어를 이용하여 H.264 복호화기의 역 정수 변환 과정과 역 양자화 과정을 고속으로 처리 할 수 있는 방법을 제안 한다. 제안하는 고속 역 변환 방법을 ZERO 블록에 대하여 역 변환과 역 양자화 과정을 수행 하지 않음으로써 속도 향상을 얻을 수 있다. 움직임이 적은 Akiyo 영상에서는 QP=0일 때 참조 코드(reference code)의 역 정수 변환과 역 양자화 과정에 비하여 7.52배, QP=24인 경우 8.1배의 속도 향상을 얻을 수 있다. 또한 움직임이 많은 Stefan 영상에 대해서는 QP=0일 때 고속 역 변환 방법이 참조 코드의 역 정수 변환과 역 양자화 과정에 비하여 6.7배, QP=36인 경우 7.83배의 속도 향상을 얻을 수 있다.

1. 서론

영상 신호와 관련하여 JPEG, MPEG, H.264과 같은 표준들은 블록 기반의 영상 압축 방식을 채택하고 있다. 상기한 표준에서는 영상신호내의 공간적 중복을 줄이기 위해 각 블록에 2D DCT을 적용한 후 고주파 성분을 제거하는 방식으로 압축 효과를 얻는다. 따라서 보편적으로 사용되는 영상 및 비디오 데이터 압축방법의 핵심 모듈인 2D DCT를 효율적으로 수행하기 위한 많은 알고리즘이 제안되었다. 2D DCT 방법에는 1D DCT를 두 번 수행하는 방법과 2D에 대해 직접 적용하는 방법이 있다 [1][2].

요즘에 쓰이는 대부분의 마이크로프로세서는 멀티미디어 응용 프로그램의 빠른 실행을 돕기 위한 멀티미디어 명령어를 포함하고 있다. 예를 들면 인텔의 제온과 펜티엄4 및 AMD의 에슬론 64는 SIMD(Single-Instruction-Multiple-Data)모델인 MMX , SSE및 SSE2 명령어들을 모두 포함하고 있다. 이러한 명령어들은 하나의 명령으로 동시에 여러 데이터를 처리 할 수 있게 해준다. 일반적으로 SIMD 연산을 위해 데이터가 미리 정렬되어 있으면 일반 명령어를 이용하는 것보다 좋은 성능을 얻을 수 있다. 그러나 SIMD 연산을 위한 데이터들이 정렬 되어 있지 않다면, 그만큼 메모리를 많이 접근해야 하기 때문에 성능의 저하를 가져온다. 심지어 SIMD 연산 과정에 사용되는 시간보다 레지스터(Register)에 데이터를 적절히 포장(Packing)하거나 해체(Unpacking)하는 과정에 소요되는 시간이 더 큰 경우가 많다 [3].

본 논문에서는 SIMD 명령어를 지원하는 프로세서에서 4x4 크기의 블록에 대한 고속 역 변환방법을 제안 하였다. 제안하는 방법은 버터플라이 표현식의 "곱셈기가 없는"(Multiplier free)특성을 이용하여 SIMD명령어 구조에 최적화 시켰으며, QP 값에 따른 양자화된 계수 값이 모두 "0"인 4x4 블록(ZERO 블록)이 나올 비율을 적용하여 역 과정을 수행 하지 않음으로써 추가적인 속도 향상을 얻을 수 있었다. ZERO 블록이 아닌 경우에도 SIMD 명령어 구조에 최적화 하였다.

2. H.264의 4x4 역 변환 방법

H.264는 주파수 영역의 데이터를 시공간 영역으로 변환하거나, 시공간의 영역의 데이터를 주파수 영역으로 변환할 때 4x4 DCT/IDCT(inverse DCT)를 사용하는 대신 4x4 정수 변환 방법을 사용한다.

식 (1)은 4x4 입력 X 에 대한 4x4 DCT를 나타낸다.

$$Y = AXA^T$$

$$= \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} \begin{bmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{bmatrix} \begin{bmatrix} a & b & a & c \\ a & c & -a & -b \\ a & -c & -a & b \\ a & -b & a & -c \end{bmatrix} \quad (1)$$

여기에서 $a = 1/2$, $b = \sqrt{1/2}\cos(\pi/8)$, $c = \sqrt{1/2}\cos(3\pi/8)$ 의 값을 가진다.

식 (1)을 인수분해 하면 식 (2)를 얻을 수 있다.

$$Y = (CXC^T) \otimes E$$

$$= \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & d & -d & -1 \\ 1 & -1 & -1 & 1 \\ d & -1 & 1 & -d \end{bmatrix} [X] \begin{bmatrix} 1 & 1 & 1 & d \\ 1 & d & -1 & -1 \\ 1 & -d & -1 & 1 \\ 1 & -1 & 1 & -d \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix} \quad (2)$$

여기에서 E는 스케일링 요소(Scaling Factor) 행렬이며, $\otimes$ 는 CXC^T 와 E 행렬의 같은 위치의 값을 서로 곱하는 기호이다. 그리고 $d = c/b \approx 0.414$ 를 갖는 상수이다. 위의 행렬식을 정수 연산으로 처리 하기 위하여 $d = 0.5$ 로

가정하면 (3)의 수식으로 간단히 할 수 있다.

$$Y = (CXC^T) \otimes E$$

$$= \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} [X] \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \\ a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \end{bmatrix} \quad (3)$$

식 (3)에서 a,b,d의 값은 $1/2, \sqrt{2/5}, 1/2$로 재 조정된다. $(C_f X C_f^T)$ 는 H.264에서 사용되는 정 방향 정수 변환의 수식을 나타낸다. 마찬가지로 기저 행렬 C는 직교하므로 $C^T C = I$ 의 특성을 가진다. 따라서 역 정수 변환 수식은 식 (4)와 같다.

$$X' = C_i^T (Y \otimes E_i) C_i$$

$$= \begin{bmatrix} 1 & 1 & 1 & d \\ 1 & d & -1 & -1 \\ 1 & -d & -1 & 1 \\ 1 & -1 & 1 & -d \end{bmatrix} \left([Y] \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix} \right) \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & d & -d & -1 \\ 1 & -1 & -1 & 1 \\ d & -1 & 1 & -d \end{bmatrix} \quad (4)$$

식 (4)에서 d는 1/2를 갖는 상수이다. 식 (3), (4)의 앞,뒤 행렬은 ±1, ±2, ±1/2 의 상수 값만을 가지고 있으며, 이 값들은 덧셈, 뺄셈, 쉬프트(shift) 연산으로 계산할 수 있다. 때문에 계산과정에서 곱셈 연산이 필요 없는 "곱셈이 없는 방법"이라고 부른다. 이 방법은 H.264의 참조 복호화기에서 효율적으로 이용되고 있다 [4].

3. 제안하는 고속 역 변환 방법

H.264는 고화질과 저비트율을 위하여 16x16 크기 블록뿐만 아니라 4x4 블록의 크기까지 총 7가지의 블록 크기가 존재 하며, 각 블록 크기에 따른 다양한 코딩 방법이 존재 한다. 또한 인터 프레임에 대한 모션 보상 방법뿐만 아니라 인트라 프레임에 대해서도 9가지 방법에 대한 인트라 예측을 통하여 4x4 블록에 대한 DC 성분을 많이 제거 하였다. 게다가 참조 프레임 (reference)에 대한 화질 보상을 위하여 Loop-filter를 포함하여 좋은 참조 프레임을 취할 수 있다 [5]. 이로써 정수 변환을 위한 4x4 입력 블록의 값들이 대부분 "0"이나 "0"에 가까운 분포하게 된다. 이런 분포를 갖는 입력 블록을 정수 변환하게 되면 저주파의 값보다는 고주파에 대부분의 값들이 존재하며, 양자화 과정을 수행하면 대부분의 양자화된 값들이 "0"이나, "1", "-1"과 같은 작은 값을 갖게 된다. 이러한 현상은 양자화 파라미터(QP) 값에 밀접하게 관계된다. 그림 1은 QP에 따른 ZERO 블록의 비율이다. 결과를 통해 알 수 있듯이 H.264는 4x4 양자화 계수에 대하여, Akiyo와 같이 움직임이 작은 영상에서는 QP에 따라 최소 50%, 최대 98%의 양자화된 계수 값이 모두 "0"인 4x4 블록(ZERO 블록)을 갖는다. 심지어 영상이 많은 Stefan의 경우에도 QP=30정도에서는 ZERO 블록의 비율의 거의 80%이상을 차지 한다. 이러한 특성을 이용하면 복호화 과정에서 양자화 계수 값에 대한 정수 변환과 역 양자화 과정 수행할 필요가 없다.

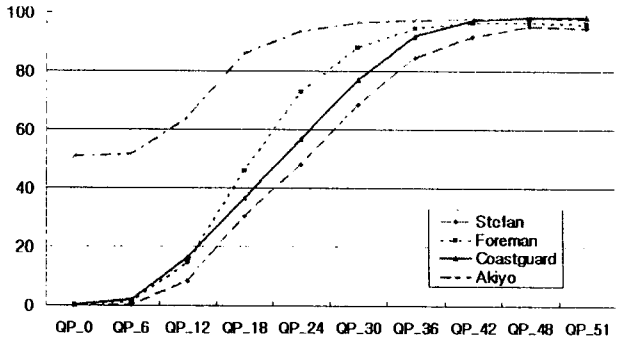


그림 1. 영상과 QP에 따른 ZERO블록 비율

이를 이용하여 제안하는 고속 역 변환 방법은 그림 2와 같이 크게 3가지 단계로 나눠 질 수 있다. 1) 입력되는 양자화 계수 값에 대한 ZERO 블록을 검사 하는 단계, 2) 4x4 역 정수 변환 하는 단계, 3) 에러 값에 대한 보상과 역 양자화 과정이다.

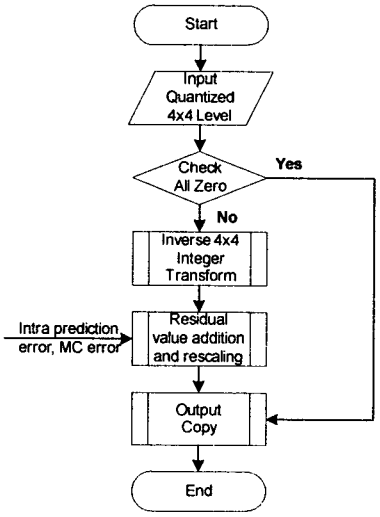


그림 2. 고속 역 변환 방법의 전체 구조

첫 번째는 입력되는 블록에 대한 ZERO블록을 검사하는 단계이며, 그림 3에 나타내었다. 4x4 블록의 데이터에 대해 ZERO블록을 검사 하기 위해서는 64비트 레지스터에 대한 총 4번의 비교 연산이 필요하다. 하지만 ZERO 블록의 확률이 매우 높기 때문에 제안하는 방법에서는 4번의 비교 연산을 수행하지 않고, 4개의 64비트 레지스터에 대한 3번의 묶인 OR(POR)연산과 64비트 결과에 대한 PSADBW 연산, 마지막으로 한번의 16비트 데이터에 대한 비교 연산으로 ZERO 블록을 검사 하였다. PDADBW 명령어는 64비트 레지스터의 각 16비트 단위로 모두 더하는 연산을 수행한다 [6]. 입력 블록의 값이 모두 "0"의 값일 경우 마지막 비교 연산의 결과는 zero 플래그를 "1"로 만들어 두 번째, 세 번째 과정을 수행하지 않게 하며, 그 외일 경우는 "0"으로 만든다. 그림 3에서 x0p는 0번째 열에 대한 4개의 데이터가 64비트 레지스터에 묶인 것을 나타낸다.

두 번째 단계는 ZERO 블록이 아닌 경우에 대한 역 정수 변환을 수행하는 과정이다. 기존의 SIMD 명령어 구조를 이용하여 정수 변환을 수행하는 방법에는 1) 2D 행렬

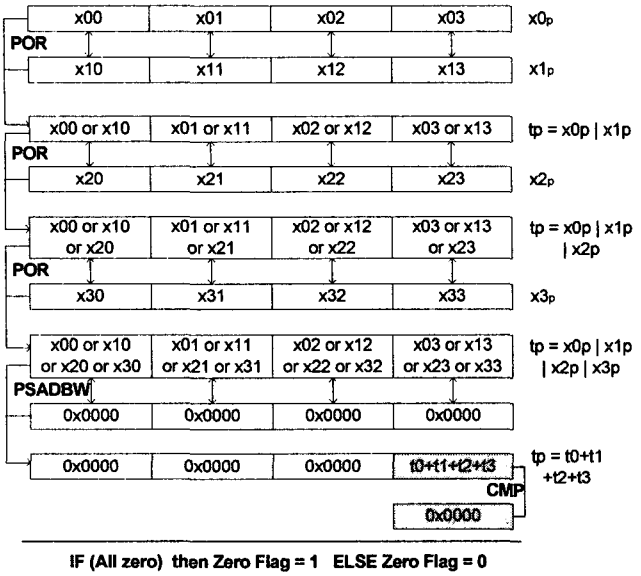


그림 3. ZERO 블록 검사 방법

곱셈 연산을 적용하는 방법, 2) 그림 4와 같은 버터플라이 표현식을 이용하는 방법, 3) 4개의 행렬을 병렬로 처리 하는 방법이 있다 [7-9]. Zhou와 그의 동료들은 버터플라이 표현식이 "곱셈이 없는" 특징을 가지고 있지만 SIMD 명령어로 병렬 처리 할 때 각각의 피 연산자가 서로 다른 연산을 수행하기 때문에 최적화 하기 어렵다고 말하고 있다. 따라서 그들은 행렬곱셈(matrix multiplication)을 이용하여 4x4 정수 변환을 수행하는 방법을 제안 하였다 [3]. Zhou의 방법은 곱셈기가 없는 프로세서에서는 곱셈 연산을 다른 명령어들로 대체 해서 사용해야 하므로 성능이 떨어 진다.

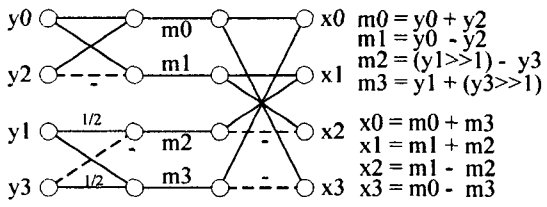


그림 4. 역 정수변환에 대한 버터플라이 표현

제안하는 방법에서는 Zhou와 버터플라이 표현식의 문제점을 해결하기 위하여 4x4 입력에 대하여 1D DCT를 수행하기 전에 먼저 전치하는 방법을 취하였다. 입력에 대하여 전치과정을 수행 하였기 때문에 피 연산자들은 동일한 연산을 동시에 수행 할 수 있다. 2D 4x4 역 정수변환의 구조는 총 4개의 단계를 갖는다. 1) 그림 5과 같이 SIMD 구조로 최적화된 4x4 블록에 대한 전치과정, 2) 그림 6과 같이 SIMD 명령어들을 이용하여 열 방향에 대한 1D 정수 변환을 수행하는 단계, 3) 1D 정수 변환의 결과에 대한 전치 단계, 4) 행 방향에 대하여 SIMD 명령어들을 이용하여 1D 정수 변환을 수행하는 단계이다.

그림 5는 역 정수 변환의 첫 번째 단계인 입력블록에 대한 전치과정이다. 이 단계에서는 PUNPCKx 명령어를 이용하여 두 개의 64비트 레지스터의 데이터의 슬롯을 삽입함으로써 전치 시킬 수 있다.

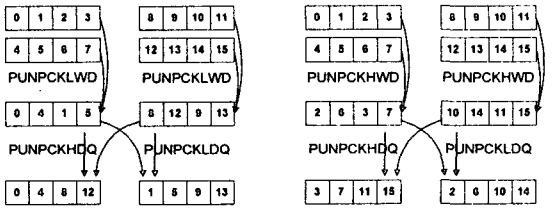


그림 5. 4x4 블록의 전치(Transpose) 과정

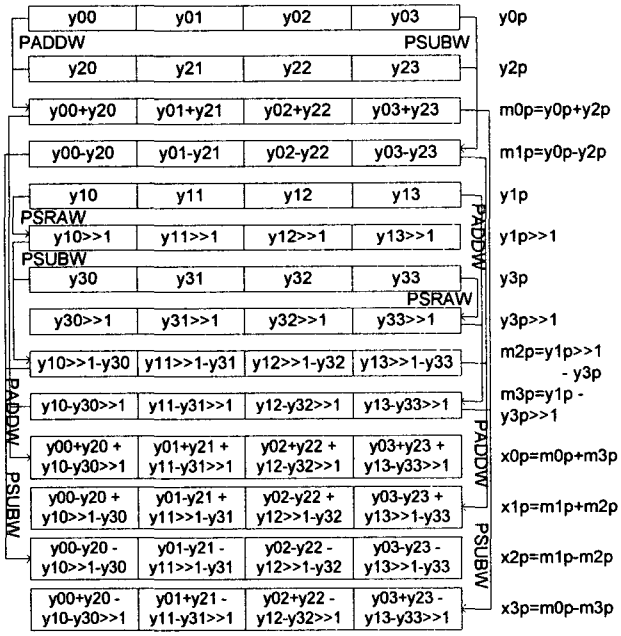


그림 6. 4x4 역 정수변환 핵심부분에 대한 SIMD 구현

두 번째 단계로는 전치된 4x4 입력에 대하여 4x4 정수 변환을 수행하는 과정이다. 4x4 역 정수 변환은 과정은 버터플라이 표현식을 SIMD 명령어들로 수행한다. 그림 6은 제안하는 역 정수 변환의 구조이다. 16비트의 데이터 4개가 묶인 형태인 y0p, y1p, y2p, y3p를 입력으로 하여 역 정수 변환을 수행한다. 이때 입력은 첫 번째 단계에서 전치과정을 수행했기 때문에 열과 행이 바뀐 상태가 되어 각각의 요소(y00, y10, y20, y30)들은 서로 같은 연산을 병렬로 동시에 수행 한다. 버터플라이 표현식의 덧셈, 뺄셈, 쉬프트 연산은 IA-32 인텔 구조 [6]에서 제공되는 PADDW, PSUBW, PSLLW의 SIMD 명령어로 바꿔어 연산을 수행하게 된다. 버터플라이 표현식의 전 과정을 수행하면 4x4 블록에 대한 1D 역 정수 변환의 수행된 결과를 얻을 수 있다.

세 번째 단계는 1D 역 정수 변환된 결과를 다시 첫 번째 단계와 마찬가지로 전치 과정을 거친 후 마지막 단계인 행 방향에 대하여 그림 6의 방법을 한번 더 수행하면 4x4 블록에 대한 2D 역 정수 변환을 수행한 결과를 얻을 수 있다.

고속 역 변환 방법의 세 번째 단계에서는 역 정수 변환의 결과에 대한 에러 보상과정이다. 식 (5)는 H.264 참조 복호화기의 C 레벨에서의 에러 보상에 대한 수식이다.

$$X_{ij} = \max(0, \min(255, W_{ij}' + (e_{ij} << 6) + 32) >> 6) \quad (5)$$

여기에서 W_{ij}'는 역 정수 변환된 계수 값이며, e_{ij}는 에러 값을 의미 한다. H.264의 역 양자화 과정에서는 4x4 역 정수 변환 과정중의 정수 연산에 의한 반올림의 오류를

줄이기 위하여 계수 값에 64가 곱해진 상태이다. 그렇기 때문에 e_{ij} 에도 64값을 곱하여 역 정수변환 계수 값과 오류 값을 더해 주어야 한다. 때문에 $e_{ij} << 6$ 과정과 반올림에 대한 오류를 줄이기 위해 32를 더해주며, 보상 값에 대한 6비트의 오른쪽 쉬프트 연산이 필요 하다. 오류 보상을 한 값은 화소 데이터로 0과 255 사이의 값이므로 min, max연산을 통하여 조절을 한다. 그림 7은 64 비트 SIMD 명령어 구조에 최적화한 에러 보상 방법이다 [4].

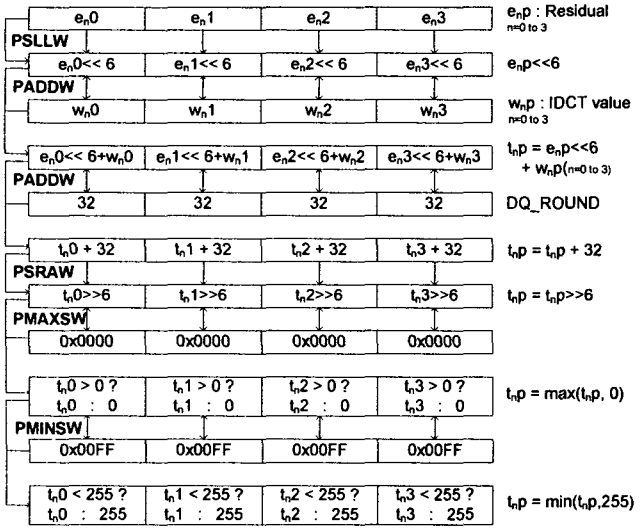


그림 7. SIMD 구조의 오류 보상 방법

4. 실험 및 결과 고찰

실험에 사용된 프로세서는 인텔 펜티엄 IV 2.4로 64비트 SIMD 명령어인 MMX를 지원한다. 실험은 Foreman, Akiyo, Stefan, Coastguard의 CIF 300 프레임을 DMB용 인코더(n=15)를 이용하여 인코딩한 후에 H.264 참조 복호화기와 제안하는 방법으로 복호화 하면서 4x4 역 정수 변환과 역 양자화 과정에서의 시간을 측정하였다. 표 1은 H.264 참조 복호화기인 JM8.2의 방법으로 복호화하여 얻은 결과이다. 300프레임에 대한 역 정수 변환과 역 양자화 과정에 걸리는 시간은 평균 2240ms정도이며, Coastguard인 경우 최대 2274ms, 최소 2231ms를 가지며 전체에 대한 차이의 비율은 1.92%이다. JM의 방법에서는 ZERO블록에 대한 아무런 이득이 없다

표 1. JM방법에 대한 영상, QP별 복호 수행시간(ms)

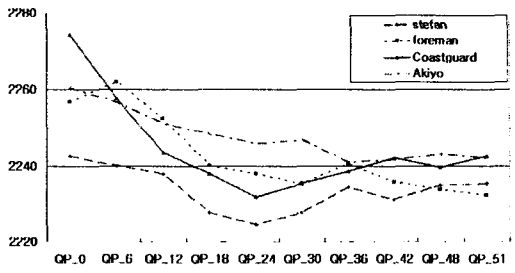


표 2는 제안하는 고속 역 변환 방법을 통하여 얻은 결과이다. 제안하는 방법은 QP에 따라서 수행시간이 감소하는 것을 볼 수 있으며, Coastguard 영상에 대하여 QP=0인 경우 333ms로 JM방법에 비하여 6.83배 빠르며, QP=36 일 경우 8.0배 정도 속도가 빠르다. (최대-최소)/최대의 비율값도 20%정도로 QP에 따라서 속도 향상을 얻을수 있다. 즉 영상에 대한 ZERO블록을 이용함으로써 20% 정도의 성능 향상을 얻었다.

표 2.고속 역 변환에 대한 영상, QP별 복호수행시간(ms)

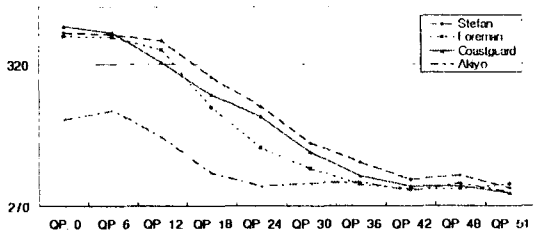
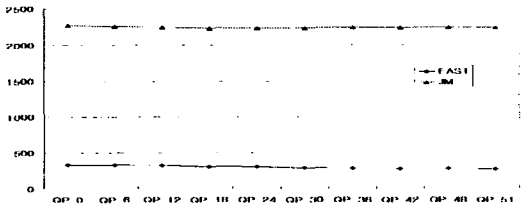


표 3.은 Coastguard 영상에 대한 JM과 제안하는 방법의 수행시간을 보여준다.

표 3. QP에 따른 Coastguard영상의 복호 수행시간(ms)



5. 결론

본 논문에서는 H.264의 복호화기에 사용되는 역 변환 방법에 대하여 SIMD 명령어를 이용하여 최적화 시키는 방법을 제안하였다. 제안 하는 방법은 ZERO 블록을 이용하여 역 변환과 역 양자화 과정을 수행하지 않음으로써 H.264의 참조 복호화기에 비해 약 8배 이상의 속도 향상을 얻을 수 있었다. 또한 제안하는 역 변환 방법에서는 곱셈기를 사용하지 않으므로 곱셈기가 없는 프로세서에서 효율적으로 사용할 수 있다.

감사의 글

본 연구는 한국 과학재단 목적기초 연구(R01-2002-000-00179-0) 과제로 수행 되었음.

참고 문헌

[1] Iain E. G. Richardson, *Video Codec Design-Developing Image and Video Compression Systems*, John Wiley & Sons Ltd, England, 2002

[2] Iain E.G Richardson, *H.264 and MPEG-4 VIDEO COMPRESSION*. WILEY, 2003

[3] X. Zhou, Eric Q. Li and Yen-Kuang Chen, "Implementation of H.264 Decoder on General-Purpose Processors with Media Instructions", *Proceeding of SPIE conference on Image and Video Communication and Processing*, volume 5022, January, 2003, pp. 224-235

[4] Iain E., G Richardson, "H.264 White paper - White Papers describing aspects of the new H.264/MPEG-4 Part 10 standard", May 12. 2004

[5] Thoms Wiegand, Gary J. Sullivan, and Ajay Luthra, "Overview of the H.264/AVC Video Coding Standard", *IEEE TRANS. on Circuits and Sys. for Video technology*. VOL13. NO.7 July 2003, pp560 - 576

[6] Intel Corp.,"IA-32 Intel® Architecture Optimization Reference Manual", Order Number: 248966-010 Intel Corp., "Intel Pentium 4 and Intel Xeon Processor Optimization - Reference Manual", Order Number: 248966-05, 2002

[7] Intel Corp., "Using Streaming SIMD Extensions in a Fast DCT Algorithm for MPEG Encoding", version 1.2, January 1999

[8] Intel Corp., "Using Streaming SIMD Extensions 2(SSE2) to Implement an Inverse Discrete Cosine Transform", vern 2.0, July 2000

[9] Intel Corp., "Streaming SIMD Extensions - Matrix Multiplication", AP-930, June 1999