논문 2004-41SP-5-13

# 공간 영역과 DCT 영역에서 MPEG-2 로부터 MPEG-4 로 변환하는 압축기의 구현

## (MPEG-2 to MPEG-4 Transcoders in The Spatial Domain and The DCT Domain)

염 인 선*, 박 현 욱**

( Insun Yeom and HyunWook Park )

요 약

멀티미디어 정보의 보급이 갈수록 확대 되어가는 요즘, 서로 다른 표준간의 음성 및 영상 데이터를 네트워크를 통해 전송할 때 이종 네트워크간의 상호 운용이 중요한 이슈가 되고 있다. 변환 압축이라는 기술이 바로 이런 문제를 해결해 줄 수 있다. 변환 압축이란 어떤 표준으로 부호화된 신호를 다른 표준의 부호화된 신호로 전환하는 기술을 뜻하며 여기에는 비트 발생량, 공간 해상도, 시간 해상도 또는 압축 표준의 변환을 모두 포함한다. 본 논문에서는 MPEG-2에서 MPEG-4로의 변환 압축기를 공간 영역과 DCT 영역에서 각각 구현하여 비교하였다. 이 변환 압축기는 디지털 방송, DVD 또는 위성방송용으로 제작된 비디오 시퀀스를 모바일 환경에서 서비스를 받을 때 유용하다. 각 표준이 지원할 수 있는 최적화된 공간 해상도를 고려하여 다운 샘플링 과정도 추가되었다. 구현된 2개의 변환 압축기의 공정한 비교를 위하여 구조의 특성상 다를 수밖에 없는 다운 샘플링 과정과 움직임 보상을 제외한 모든 블록에 대해서 동일하게 구현하였으며 빠른 변환 압축 부호화가 이루어지도록 움직임 추정을 다시 하지 않고 복호된 정보를 그대로 사용하였다. 결과 영상의 화질 비교와, 출력 파일의 크기 그리고 수행 시간에 대하여 공간 영역에서 구현한 변환 압축기가 DCT 영역에서 구현한 변환 압축기보다 더 나음을 확인하였다.

**Abstract**

Various multimedia systems have been developed and their application areas widely proliferate. Thus, the interoperability is getting important among various networks and devices. The video transcoding is a technology to solve this interoperability problem among various coding standards. Transcoding can be defined as the conversion of one compressed coded data to another. In this paper, MPEG-2 to MPEG-4 transcoder in the spatial domain is compared with that in the DCT domain. The transcoder is very useful when a video sequence that is originally encoded for digital TV, DVD or satellite broadcasting is served in mobile environment. In order to compare two transcoders, all modules except motion compensation and down sampling are implemented identically. In addition, both transcoders do not search for motion vector. Instead, the decoded information is reused to the encoder. The experimental results show that the transcoder in the spatial domain is usually better than that in the DCT domain with respect to PSNR (Peak Signal-to-Noise Ratio), bitrate and execution time.

**Keywords:** 변환 압축기 (Transcoder), 움직임 보상 (Motion Compensation), 다운 샘플링 (Down Sampling)

## I. 서 론

멀티미디어의 발전은 음성 및 영상 데이터의 효과적인 전송과 저장을 요구하게 되었고, 다양한 해상도와 표준을 지원하는 시퀀스들을 네트워크를 통해 전송할 때 이종 네트워크간의 호환이 중요한 이슈가 되었다. 이런 문제는 어떤 표준으로 압축된 정보를 다른 표준의 압축된 정보로 바꾸어 주는 변환 압축 부호화라는 방법으로 해결이 가능하다[1]-[2]. 초기의 변환 압축 부호화는 발생 비트량을 수신측의 대역폭에 맞추어 줄이는 것에 초점을 맞추었지만 그 후에 프레임을 적절히 줄이거나 다운 샘플링을 통하여 공간 해상도를 줄이고 압축 표준

* 학생회원, ** 정회원, 한국과학기술원 전자전산학과
(Dept. of Electrical Engineering and Computer Science, KAIST )
접수일자: 2004년4월28일, 수정완료일: 2004년7월7일

을 바꾸는 것으로 점점 그 개념이 확대되어 갔다. 요즘은 데이터를 전송할 때 발생할 수 있는 에러에 대처하는 변환 압축기의 연구도 진행되고 있다[2].

본 논문에서는 MPEG-2로 부호화된 입력 비트 스트림을 복호화한 후 다운 샘플링을 거쳐 공간 해상도를 절반으로 줄인 후 MPEG-4로 다시 부호화하는 변환 압축기를 공간 영역에서 이루어지는 것과 DCT 영역에서 이루어지는 것을 구현하여 비교하였다. Ⅱ장에서는 공간 영역에서의 변환 압축기를, Ⅲ장에서는 DCT 영역에서의 변환 압축기를 설명하며 Ⅳ장에서 두 변환 압축기를 비교한 실험 결과를 제시하며 Ⅴ장에서 결론을 맺는다.

## Ⅱ. 공간 영역에서 구현한 변환 압축기

그림 1은 공간 영역에서 구현한 MPEG-2에서 MPEG-4로 변환되는 압축기의 블록 다이어그램이다. 변환 압축기의 입력은 MPEG-2로 부호화된 CIF (Common Intermediate Format: 352×288) 사이즈의 비트 스트림이며 이것이 MPEG-2로 복호된다. 다운 샘플링은 선형 보간을 통하여 이루어진다. 이 후 QCIF (Quarter CIF: 176×144) 사이즈의 시퀀스가 MPEG-4로 부호화된다. 기존의 변환 압축기와 달리, 두 번의 IDCT와 한 번의 DCT 과정을 빠르게 수행할 수 있도록 Chen 알고리듬으로 대체하였다[3]. 이 알고리듬을 이용하면 실수형의 연산을 정수형의 연산으로 대체할 수 있으며 상당한 수

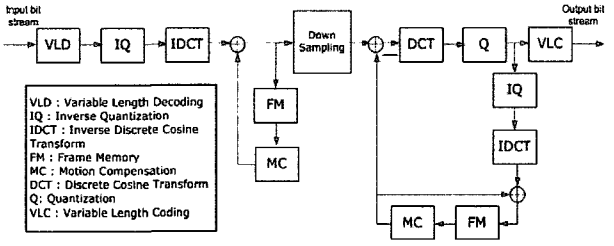



그림 1. 공간 영역에서 구현한 변환 압축기
Fig. 1. MPEG-2 to MPEG-4 transcoder in the spatial domain.

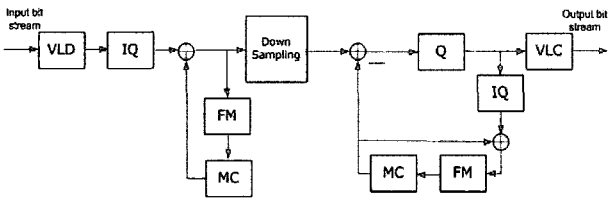



그림 2. DCT 영역에서 구현한 변환 압축기
Fig. 2. MPEG-2 to MPEG-4 transcoder in the DCT domain.

의 곱셈의 개수를 줄일 수 있으므로 수행 시간이 단축된다.

## Ⅲ. DCT 영역에서 구현한 변환 압축기

그림 2는 DCT 영역에서 구현한 MPEG-2에서 MPEG-4로 변환되는 압축기의 블록 다이어그램이다. 모든 변환 압축 과정이 DCT 영역에서 이루어지므로 DCT와 IDCT 과정이 필요 없다. 또한 다운 샘플링도 공간 영역처럼 선형 보간을 하지 않고 DCT 영역에서 직접 이루어지게 하였다. DCT 영역에서 구현한 변환 압축기의 특성상, 움직임 보상과 다운 샘플링이 공간 영역에서의 변환 압축기와 다를 수밖에 없다. 다음 2개의 소단락은 움직임 보상과 다운 샘플링 과정을 설명한다.

### 1. DCT 영역에서의 움직임 보상

공간 영역에서의 움직임 보상은 움직임 벡터의 크기만큼 떨어져 있는 곳의 값들을 그대로 가지고 오거나 또는 보간을 통해서 쉽게 얻을 수 있다. 하지만 DCT 영역에서는 대부분의 움직임 벡터가 수평, 수직 방향으로 동시에 정렬이 되어 있지 않기 때문에 같은 방법으로 움직임 보상을 할 수가 없다. 공간 영역에서의 움직임 보상은 행렬 연산을 통해서도 가능하다 [식 1, 식 2, 그림 3]. 각 매크로블록을 4개의 8 × 8 블록으로 세분화시킨 후 각 블록의 앞뒤로 수직, 수평 방향으로 이동시키는 행렬을 곱함으로써 한 매크로블록에 대한 움직임 보상을 할 수 있다. 그림 3을 설명한 식이 식 1과 식 2이다. 수평, 수직 방향으로 이동시키는 행렬은 그림 3처럼 참조 블록이 얼마나 걸쳐 있는가에 따라 달라지며, 그 크기 만큼에 해당하는 단위 행렬을 가지고 나머지는 0으로 채워진다.

$$P_{ref} = \sum_{i=1}^{4} H_{i1} P_i H_{i2} \tag{1}$$

$$H_{i1} = \begin{bmatrix} 0 & 0 \\ I_H^i & 0 \end{bmatrix}, \; H_{i2} = \begin{bmatrix} 0 & I_W^i \\ 0 & 0 \end{bmatrix} \tag{2}$$

DCT 영역의 움직임 보상은 식 1의 각 행렬에 DCT를 한 값들의 곱으로 얻을 수 있다. 공간 영역의 8 × 8 블록의 값들은 DCT 계수 값들로 바뀔 것이고 수직, 수평 방향으로 이동시키는 행렬도 DCT를 한 값들이다[4]. 이 값들은 모든 경우의 수에 대해서 미리 값들을 구

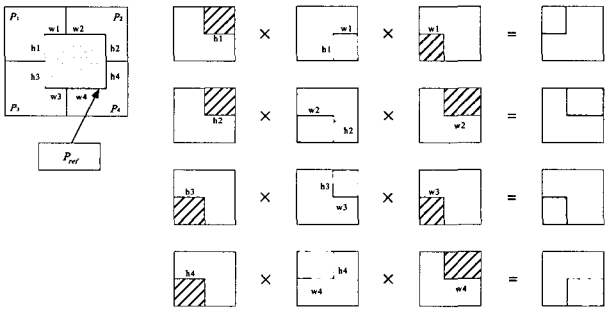


그림 3. 행렬 연산을 이용한 움직임 보상
Fig. 3. Motion compensation using matrix operation.

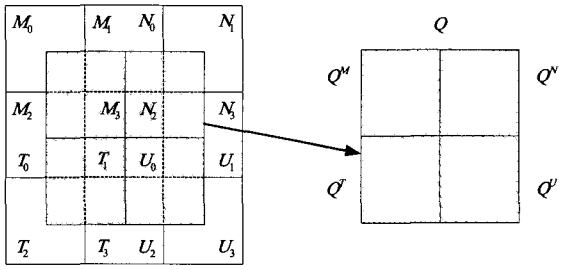


그림 4. 한 매크로블록에 대한 예측
Fig. 4. Prediction of a 16x16 macroblock.

해서 메모리에 저장하도록 하였다.

DCT 영역에서 이루어지는 움직임 보상을 좀 더 빨리 할 수 있는 여러 알고리듬이 있는데[5]-[6], 화질의 저하 없이 단지 행렬 연산을 위한 입력 블록을 다시 재배열 하여 행렬의 곱을 줄일 수 있는 알고리듬을 이용하여 빠른 움직임 보상을 이루도록 하였다[7]. 그림 4에서 움직임 보상을 해야 할 매크로블록 $Q$는 4개의 블록인 $Q^M$, $Q^N$, $Q^T$, $Q^U$로 이루어져 있다. 각각의 8× 8 블록에 대한 움직임 보상은 식 1과 식 2에 따라 전개할 수 있다. 행렬을 이용한 움직임 보상은 세 가지의 특징을 가지고 있다.

첫째, 식 1처럼 각 8× 8 블록에 필요로 하는 $P_i$의 개수는 4개이고 한 개의 매크로블록에 필요한 $P_i$의 개수는 16이지만 실제 움직임 보상에 필요한 8× 8 블록의 개수는 그림 4처럼 9개에 불과하다.

둘째, 그림 3에 나와 있는 것처럼 수평, 수직 방향으로 겹쳐 있는 양이 같다는 것이다. 즉 $h1=h2$, $h3=h4$, $w1=w3$, $w2=w4$ 이고 나머지 3개의 8× 8 블록에서도 똑같이 적용된다.

마지막으로, 순열 행렬 (Permutation matrix) 의 도입이다. 식 1과 식 2에서 정의한 $H$ 행렬에서 $H_{11}+H_{31}=P^1$, $H_{12}+H_{22}=P^0$ 로 정의할 때 $P$는 순열 행렬이 된다.

이 세 가지의 특성을 이용하여 $Q^M$, $Q^N$, $Q^T$, $Q^U$에 관한 식을 정리하면 다음과 같다.

$$Q^M=\sum_{i=0}^{3}Q^{M_i}$$
$$=H_{11}(M_0-M_1-M_2+M_3)H_{12}+H_{11}(M_1-M_3)P^0+P^1(M_2-M_3)H_{12}+P^1M_3P^0 \quad (3)$$

$$Q^N=\sum_{i=0}^{3}Q^{N_i}$$
$$=H_{11}(N_1-N_0-N_3+N_2)H_{22}+H_{11}(N_0-N_2)P^0+P^1(N_3-N_2)H_{22}+P^1N_2P^0 \quad (4)$$

$$Q^T=\sum_{i=0}^{3}Q^{T_i}$$
$$=H_{31}(T_2-T_3-T_0+T_1)H_{12}+H_{31}(T_3-T_1)P^0+P^1(T_0-T_1)H_{12}+P^1T_1P^0 \quad (5)$$

$$Q^U=\sum_{i=0}^{3}Q^{U_i}$$
$$=H_{31}(U_3-U_2-U_1+U_0)H_{22}+H_{31}(U_2-U_0)P^0+P^1(U_1-U_0)H_{22}+P^1U_0P^0 \quad (6)$$

원래는 총 16개의 행렬 곱 $ABC$ (모두 8× 8 단위)가 존재하지만 식 3에서 식 6을 그림 4와 비교해서 살펴보면 공통된 항이 있음을 알 수 있다. 따라서 이 알고리듬을 이용하면 필요한 행렬 $ABC$의 곱 16개는 9개로 줄어들어 44%의 이득을 얻을 수 있다[8].

위의 경우는 움직임 벡터가 수평, 수직 방향으로 정렬되어 있지 않는 경우이다. 만약 움직임 벡터가 한쪽 방향으로만 정렬되어 있다면 원래 필요한 행렬 $ABC$의 곱은 16개 아니라 8개가 되며 이 알고리듬에 의하여 6개로 줄어들게 된다. 움직임 벡터가 모두 정렬되어 있다면 이 알고리듬을 이용한 이득은 없다.

## 2. DCT 영역에서의 다운 샘플링

움직임 보상을 통해 얻은 DCT 계수들은 다시 IDCT를 하지 않고 DCT 영역에서 다운 샘플링을 하게 된다. 그림 5는 1차원에서 다운 샘플링을 하는 과정이 두 영역에서 어떻게 이루어지는지를 보여준다[9]. 먼저 공간 영역에서 8개의 픽셀 값 2개와 그것의 DCT 계수를 생각한다. 공간 영역에서 각각의 8개의 값들 2개를 확장시키고 이것과 동일한 DCT 영역에서의 값을 고려한다 [그림 5의 (b)]. 이 과정은 DCT 계수 값에 0을 넣는 것이다. 그림 5의 (c)는 윈도우를 보여주고 있다. 공간 영역에서 확장시킨 계수와 윈도우를 곱해서 [그림 5의 (d)의 왼쪽] 더해주면, 처음의 8개의 값들 2개와 같은 결과를 얻게 된다 [그림 5의 (e)의 왼쪽]. 이와 동일한 DCT 영역에서의 연산은 대칭 Convolution을 이용하는 것이다. 바로 이 과정을 거치면 처음의 8개의 DCT 계수 2개에서 16 개의 DCT 계수 한 개를 얻을 수 있다. 여기

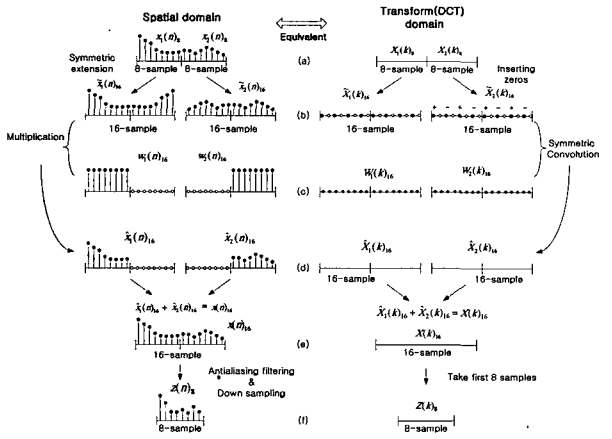



그림 5. DCT 영역에서의 다운 샘플링
Fig. 5. Down sampling in the DCT domain.

서 처음의 8개의 값을 취함으로써 일차원에서의 다운 샘플링을 하게 된다. 이 연산을 수직, 수평 방향으로 적용시킴으로써 이차원에서의 다운샘플링을 할 수 있다.

## Ⅳ. 구현한 변환 압축기들의 비교

변환 압축기 구조의 특성상, 움직임 보상과 다운 샘플링 과정은 다르다. 각 영역에서 행하는 움직임 추정 과정 역시 다를 뿐 아니라 상당한 시간이 요구되므로 복호화된 정보를 부호기로 전달하도록 하였다. 다운 샘플링 과정을 거치므로 부호기의 입력으로 들어가는 각 매크로블록마다의 움직임 벡터와 모드를 결정해 주어야 한다. 움직임 벡터는 식 7에서 보는 것처럼 거리가 가장 적은 중간값(*Median*)으로 얻었고 모드는 4개의 매크로블록 중 3개 이상이 INTRA 모드이면 INTRA로, 3개 이상이 SKIP이면 SKIP으로 그 외의 경우는 INTER 모드로 결정하였다.

$$V = \{v_1, v_2, v_3, v_4\}, d_i = \sum_{\substack{j=1 \\ j \neq i}}^{4} \mathrm{P}\, v_i - v_j \,\mathrm{P}$$
$$Median(V) = v_k \in V \text{ such that } \min(d_i) = d_k \qquad (7)$$

비교 대상은 PSNR (Peak Signal-to-Noise Ratio), 출력 파일의 크기, 수행 시간 그리고 수행 시간의 결과를 뒷받침할 수 있는 움직임 벡터의 정렬 정도와 각 영역에서 변환 압축을 할 때 필요로 하는 곱셈과 덧셈의 개수이다. 실험 대상 시퀀스는 움직임이 많은 *Foreman*, *Stefan* 시퀀스와 움직임이 별로 없는 *Td*, *Akiyo* 시퀀스에 대해서 조사하였다. 변환 압축기의 입력은 3Mbps로 부호화된 300 프레임의 비트 스트림이다.

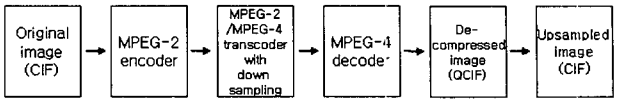
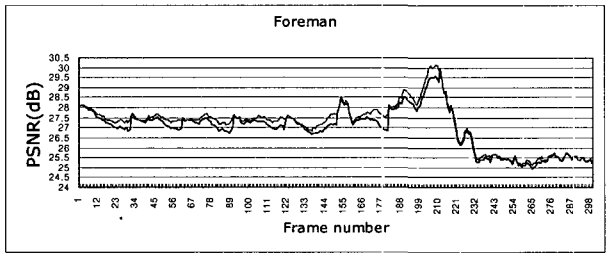
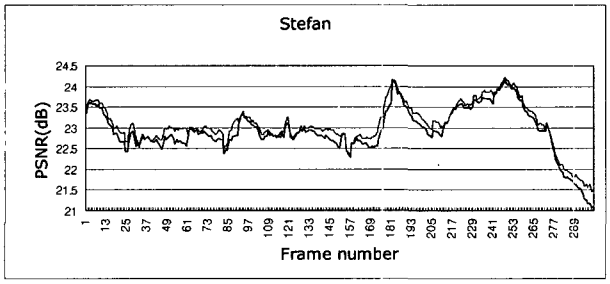


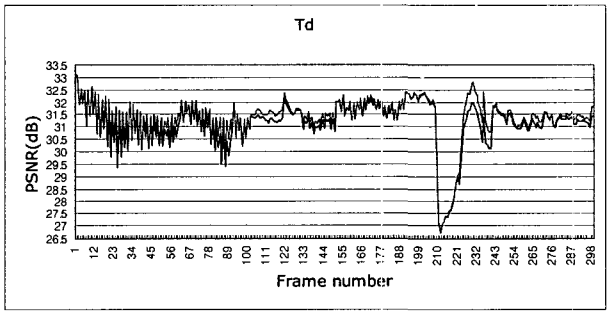

그림 6. PSNR의 대상이 되는 두 이미지
Fig. 6. Two images for PSNR.



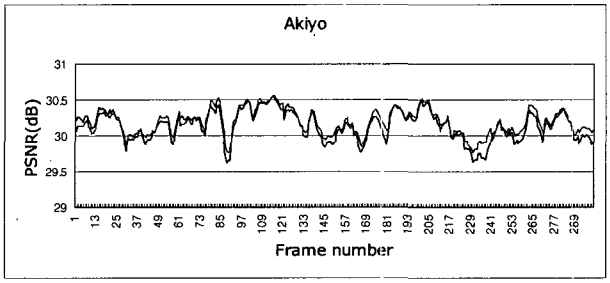

(a) *Foreman*




(b) *Stefan*




(c) *Td*




(d) *Akiyo*

그림 7. PSNR 결과
(회색: 공간 영역, 검은색: DCT 영역)
Fig. 7. PSNR Result
(Gray: spatial domain, Black: DCT domain)

### 1. PSNR

PSNR을 구하기 위해 출력 파일을 MPEG-4로 복호화한 QCIF 이미지를 그림 6에서와 같이 6-tap 필터를

써서 CIF 사이즈로 확대시킨 후 원래의 이미지와 비교하였다. DCT 영역에서 대칭 Convolution을 이용한 다운 샘플링은 공간 영역에서의 선형 보간 보다 더 좋은 필터의 특성을 나타내므로 GOP (Group of Picture)의 첫 프레임은 DCT 영역에서 구현한 것이 조금 나은 화질을 보여준다. 그러나 DCT 영역에서의 구한 움직임 보상은 Mismatch 현상이 일어나는 치명적인 단점을 가지고 있다[10]. Mismatch 현상이랑 DCT 영역에서 움직임 보상을 한 실수형의 결과를 2 Byte의 정수형으로 나타내어야 하는 데서 기인하는 정도(精度) 문제이다. 따라서 매 프레임이 들어올 때마다 움직임 보상을 할 때 필요로 하는 참조 영상의 에러가 계속 누적되어 화질의 저하가 뚜렷하게 발생한다. 따라서 그림 7에서 보는 바와 같이 대부분의 테스트 시퀀스에서는 공간 영역에서 구현한 변환 압축기가 DCT 영역에서 구현한 것보다 대부분 화질이 좋음을 확인할 수 있다. 단 *Akiyo* 같이 움직임이 거의 없는 시퀀스의 경우는 움직임 보상을 하기 위한 행렬의 곱이 거의 없기 때문에 상대적으로 Mismatch 현상이 덜 생기고 따라서 DCT 영역에서 구현한 것이 공간 영역에서 구현한 것보다 PSNR이 높은 부분도 확인할 수 있다.

## 2. 출력 파일의 크기

표 1은 출력 파일의 결과이다. 움직임이 많은 *Foreman*, *Stefan* 시퀀스의 경우 공간 영역에서 구현한 것의 출력 파일이 약 16% 적고, 움직임이 적은 *Td*, *Akiyo* 시퀀스의 경우 공간 영역에서 구현한 것의 출력 파일이 약 10% 적음을 알 수 있다.

## 3. DCT 영역에서 필요로 하는 명령어의 개수

DCT 영역에서 구현한 변환 압축기에 필요한 명령어의 개수는 움직임 보상과 다운 샘플링에 국한시킬 수 있다. 움직임 보상을 수행할 때, 행렬 연산이 필요하고 이는 대부분 8× 8 단위의 행렬의 곱셈으로 이루어져 있다. 그러나 움직임 보상을 수행할 때 모든 매크로블록에 필요로 하는 행렬의 곱셈의 개수는 움직임 벡터가 어떤 값을 가지느냐에 따라 좌우되고, 이것이 전체 수행 시간을 결정한다. 따라서 움직임 벡터의 정렬 정도가 DCT 영역에서 구현한 변환 압축기의 수행 시간에 어느 정도 비례함을 짐작할 수 있다.

표 2는 움직임 보상을 할 때 각 매크로블록에 필요한 명령어의 개수이다. 표 2에서 "Two directions aligned"는 움직임 벡터가 수직, 수평 방향으로 모두 정렬되어 있어서 행렬 연산이 필요 없는 경우이다. "One direc-tion aligned"는 움직임 벡터가 한 쪽 방향으로만 정렬이 되어 있는 경우이다. 이 경우는 수평, 수직 방향으로 이동시키는 행렬 두 개 중 한 개는 단위행렬이므로 움직임 벡터가 어느 방향으로도 정렬이 되어 있지 않은 "No aligned"의 경우보다 상대적으로 필요로 하는 명령어의 개수가 적음을 알 수 있다.

표 1. 출력 파일의 크기 (단위: KB)
Table 1. File size of output bit stream (Unit: KB).

| | DCT 영역 | 공간 영역 |
|---|---|---|
| *Foreman* | 571 | 478 |
| *Stefan* | 1,797 | 1,510 |
| *Td* | 313 | 279 |
| *Akiyo* | 95.1 | 86.8 |

표 2. DCT 영역에서 움직임 보상할 때 각 매크로블록 당 필요로 하는 명령어의 개수
Table 2. Number of instructions per macroblock that is needed to execute motion compensation in the DCT domain.

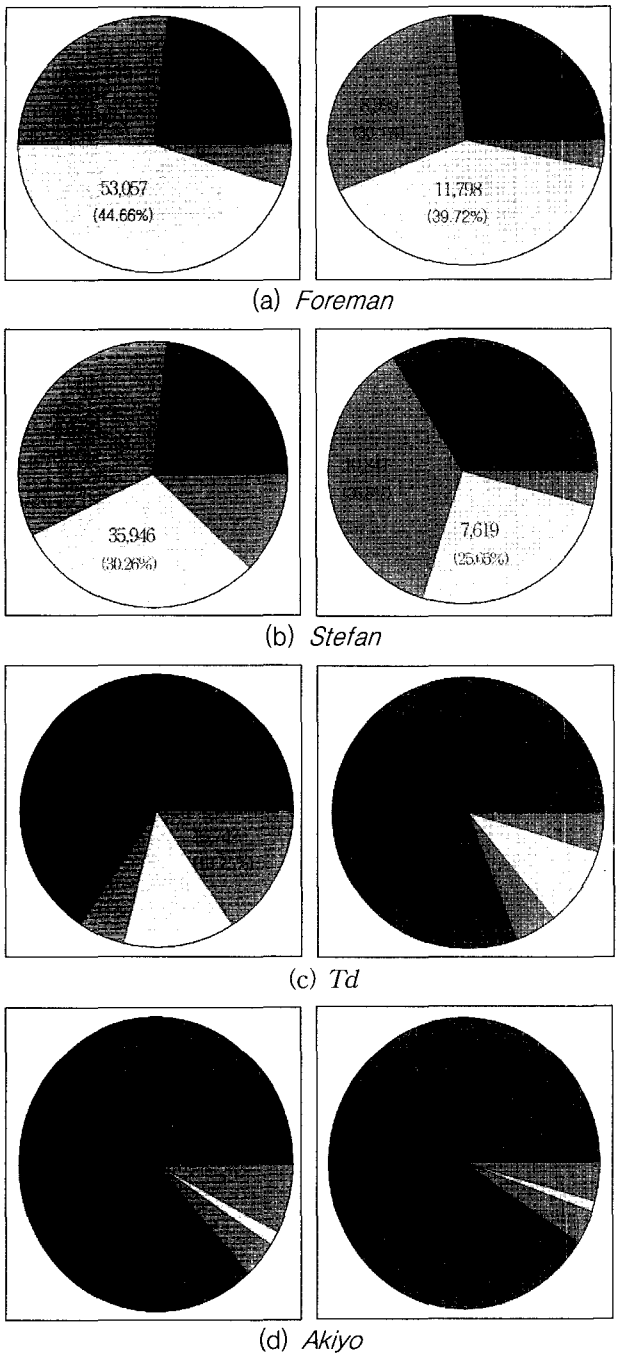
| | Two directions aligned | One direction aligned | No aligned |
|---|---|---|---|
| 곱셈 | 0 | 5,120 | 17,408 |
| 덧셈 | 0 | 5,376 | 18,496 |
| 쉬프트 | 0 | 640 | 3,200 |

그림 8은 4개의 시퀀스에 대한 움직임 벡터의 정렬 분포를 나타낸다. 복호기는 CIF 사이즈에 대한 것이므로 전체 매크로블록의 개수는 396×300=118,800 개이고, 부호기는 QCIF 사이즈에 대해 부호화 과정을 수행하므로 전체 매크로블록의 개수는 99×300=29,700개이다. 같은 시퀀스라 하더라도 모드를 새로 정해 주는 과정 때문에 인코더와 디코더의 움직임 벡터의 정렬 정도는 다소 차이가 있다.

대칭 Convolution을 이용하는 DCT 영역의 다운 샘플링은 다운 샘플링 과정을 수행하는 행렬을 미리 정할 수 있다. 그러나 이 행렬은 값이 0인 구성원이 많고 대칭이 되는 계수가 있으므로 직접 풀어서 다운 샘플링을 수행할 수 있게 하였으며 표 3에 필요로 하는 명령어의 개수를 나타내었다.

## 4. 공간 영역에서 필요로 하는 명령어의 개수

공간 영역에서 구현한 변환 압축기의 경우 두 번의




(a) *Foreman*

(b) *Stefan*

(c) *Td*

(d) *Akiyo*

그림 8. 움직임 벡터의 분포
(왼쪽: 복호기, 오른쪽: 부호기)

Fig. 8. Statistics of motion vector.
(Left: Decoder, Right: Encoder)

표 3. DCT 영역의 다운 샘플링에 필요한 명령어의 개수

Table 3. Number of instructions needed for down samp -ling in the DCT domain.

| | |
|---|---|
| 곱셈 | 456,192 |
| 덧셈 | 114,048 |
| 쉬프트 | 912,384 |

표 4. IDCT와 FDCT 에 필요한 명령어의 개수
(< >는 FDCT에 필요한 명령어의 개수)

Table 4. Number of instructions for IDCT and FDCT.
(< > indicates number of instructions for FDCT)

| | 8x8 | One frame (CIF) | One frame (QCIF) |
|---|---|---|---|
| 곱셈 | 256 <224> | 608,256 | 152,064 <133,056> |
| 덧셈 | 640 <600> | 1,520,640 | 380,160 <356,400> |
| 쉬프트 | 288 <144> | 684,288 | 171,072 <85,536> |

표 5. Bilinear interpolation에 필요한 명령어의 개수

Table 5. Number of instructions for bilinear interpolation.

| | |
|---|---|
| 덧셈 | 152,064 |
| 쉬프트 | 38,016 |

표 6. 움직임 보상에 필요한 명령어의 개수
(디코더/인코더)

Table 6. Number of instructions for motion compensation.
(Decoder/Encoder)

| | |
|---|---|
| 덧셈 | 최대 633,600/ 158,400 |
| 쉬프트 | 최대 136,224/ 34,056 |

IDCT와 한 번의 DCT, 다운 샘플링과 움직임 보상 루틴에 대한 명령어의 개수가 IV-3 에서 조사한 것과 대응이 된다. 공간 영역에서 구현한 변환 압축기는 IDCT와 FDCT에서만 곱셈이 필요하며 이 때 필요한 곱셈의 개수는 시퀀스와 무관하다 [표 4]. 선형 보간을 이용한 다운 샘플링과 움직임 보상은 덧셈과 쉬프트 연산으로 가능하므로 모든 시퀀스에 필요한 곱셈의 개수는 동일함을 알 수 있다 [표 5, 표 6]. 움직임 보상을 할 때는 움직임 벡터가 반 화소 단위를 가지는지에 따라 필요로 하는 명령어의 개수가 다르며 표 6은 모든 매크로블록이 수직, 수평 방향으로 반 화소 단위를 가졌을 때에 필요한 명령어의 개수를 나타내었다.

## 5 각 프레임당 필요로 하는 평균 명령어의 개수와 수행 시간의 비교

표 7는 IV-3과 IV-4에서 얻은 명령어의 개수를 종합하여 각 프레임 당 필요로 하는 전체 곱셈의 개수와 덧셈의 개수를 나타내었다. 그리고 표 8은 PC (Pentium

표 7. 각 프레임당 필요로 하는 명령어의 개수
Table 7. Number of instructions for one frame transco -ding.

표 7-(a). 각 프레임당 필요로 하는 곱셈의 평균 개수
Table 7-(a). The average number of multiplication for one frame transcoding.

| | 공간 영역 | DCT 영역 |
|---|---|---|
| *Foreman* | 893,736 | 4,907,520 |
| *Stefan* | 893,736 | 4,139,520 |
| *Td* | 893,736 | 1,584,432 |
| *Akiyo* | 893,736 | 680,448 |

표 7-(b). 각 프레임당 필요로 하는 덧셈의 평균 개수
Table 7-(b). The average number of addition for one frame transcoding.

| | 공간 영역 | DCT 영역 |
|---|---|---|
| *Foreman* | 2,843,381 | 4,834,944 |
| *Stefan* | 2,787,993 | 4,002,086 |
| *Td* | 2,527,024 | 1,847,808 |
| *Akiyo* | 2,439,430 | 350,410 |

표 8. 수행 시간 (단위: 초)
Table 8. Execution time(Unit: sec.).

| | 공간 영역 | DCT 영역 |
|---|---|---|
| *Foreman* | 4.8 | 9.6 |
| *Stefan* | 5.2 | 8.6 |
| *Td* | 4.6 | 5.2 |
| *Akiyo* | 4.5 | 3.9 |

4, 2.2 GHz)에서 측정한 두 변환 압축기의 수행시간 비교이다. 움직임이 거의 없는 *Akiyo* 시퀀스만 DCT 영역에서 구현한 변환 압축기가 더 빠름을 알 수 있고 이 결과는 표 7-(a)에서 보듯이 DCT 영역에서 필요로 하는 곱셈의 개수가 공간 영역에서 필요로 하는 개수보다 적다는 것이 뒷받침해준다.

## V. 결 론

DCT 영역에서 구현한 변환 압축기는 DCT와 IDCT를 수행하지 않기 때문에 공간 영역에서 구현한 것보다 더 빠를 것으로 예상했지만, 다운 샘플링과 움직임 보상에 필요로 하는 곱셈의 개수가 공간 영역에서 필요로 하는 곱셈의 개수 보다 상대적으로 많고 또한 공간 영역에서 필요로 하는 IDCT와 FDCT를 정수형으로 빠르게 처리할 수 있는 Chen 알고리듬을 적용하였기 때문에 오히려 DCT 영역에서 구현한 것이 대부분의 시퀀스에서 더 많은 시간을 필요하다는 것을 확인하였다. DCT 영역의 경우 수행시간은 움직임 벡터가 어떻게 정렬되어 있는지에 따라서 좌우된다. 공간 영역은 곱셈의 개수가 모든 시퀀스에 대해 같기 때문에 수행 시간은 파일의 크기와 밀접한 관련이 있다.

DCT 영역의 변환 압축기의 특징은 정도(精度)의 불일치에서 발생하는 Mismatch 현상 때문에 화질의 저하가 일어나고 따라서 전반적으로 공간 영역에서 구현한 변환 압축기의 성능이 DCT 영역에서 구현한 것보다 나음을 알 수 있다.

## 참 고 문 헌


[1] Niklas Bjork, "Transcoder Architectures for Video Coding," *IEEE Transactions on Consumer Electronics*, Vol. 44, No. 1, Feb. 1998.

[2] Anthony Vetro, "Video Transcoding Architectures and Techniques: an Overview", *IEEE Signal Processing Magazine*, Mar. 2003.

[3] TMS320C64x Image/Video Processing Library

[4] Wenwu Zhu, "CIF-to-QCIF Video Bitstream Down-Conversion in the DCT Domain," *Bell Labs Technical Journal*, Jul. 1998.

[5] P.A.A. Assuncao and M.Ghanbari, "Transcoding of MPEG-2 video in the frequency domain", *ICASSP 1997*, pp. 2633-2636, 1997.

[6] N. Merhav and V. Bhaskaran, "A fast algorithm for DCT domain inverse motion compensation," *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing*, Vol. 4, pp. 2307-2310, May 1996.

[7] Junehwa Song, Boon-Lock Yeo, "A Fast Algorithm for DCT-Domain Inverse Motion Compensation Based on Shared Information in a Macroblock," IEEE *Transactions on Circuits and Systems for Video Technology*, Vol. 10, No. 5, Aug. 2000.

[8] S.F. Chang and D.G. Messerschmitt, "Manipu -lation and compositing of MC-DCT compressed video," *IEEE J. Select. Areas Commun.*, Vol. 13, pp. 1-11, Jan. 1995.

[9] HyunWook Park, YoungSeo Park, Seung-Kyun Oh, "L/M-fold Image Resizing in Block-DCT Domain Using Symmetric Convolution," *IEEE Transactions on Image Processing*, Vol. 12, No. 9, Sep. 2003.

[10] Seung-Kyun Oh, HyunWook Park "Precision

lifting method to reduce the mismatches between spatial- and transform-domain motion-compensa -ted coders," *Image Processing, 2003. Procee -dings. 2003 International Conference on*, Vol. 3, Sept. 14-17, 2003.

## 저 자 소 개

염 인 선(학생회원)
2002년 한양대학교 전자전기 공학부 졸업 (공학사).
2004년 8월 한국과학기술원 전자전산학과 전기 및 전자공학전공 졸업 (공학석사)

<주관심분야: 영상압축, 트랜스 코딩>

박 현 욱(정회원)
1981년 서울대학교 전기공학과 졸업 (공학사).
1983년 한국과학기술원 전기 및 전자공학과 졸업 (공학석사).
1988년 한국과학기술원 전기 및 전자공학과 졸업 (공학박사).
1989년~1992년 University of Washington 연구원
1992년~1993년 삼성전자 수석연구원.
1993년~현재 한국과학기술원 전자전산학과 전기 및 전자공학전공 교수.
<주관심분야: 영상처리, 영상압축, 의료영상시스템, 멀티미디어 시스템>