FSM 기법을 이용한 효과적인 run_before 복원 방식

정회원 문 용 호*

An Effective Run-before Decoding Method Based on FSM

Yong-ho Moon* ***Regular Member***

요 약

일반적으로 H.264/AVC 압축 표준의 CAVLC 복원 기법에 있어서 많은 횟수의 메모리 액세스가 요구되어진다. 그런데 이것은 상당한 전력 소모를 가져오기 때문에 DMB 및 비디오폰 서비스에 있어서 큰 문제가 된다. 이러한 문제를 해결하기 위하여 본 논문에서는 효율적인 run_before 복원 방식을 제안한다. 제안 방식에서는 산술 연산으로 구성된 FSM 기법을 토대로 하여 복호화시에 야기되는 메모리 액세스가 제거된다. 모의 실험 결과는 제안 방식에 의하여 화질의 손실이 발생하지 않으며 시스템 Power가 절약됨을 보여준다.

Key Words : H.264/AVC, CAVLC, Complexity, Run-before, FSM

ABSTRACT

In general, a large number of the memory accesses are required to decode the CAVLC in H.264/AVC. This is a serious problem for applications such as a DMB and videophone services because of the considerable amount of power that is consumed in accessing the memory. In order to overcome this problem, we propose an efficient run_before decoding method, In the proposed method, the memory access is removed by using a FSM with arithmetic operations. The simulation results show that the proposed algorithm does not degrade video quality is not degraded as well as the power is saved.

Ⅰ. 서 론

DVD, 디지털 TV, 그리고 VoD 등과 같은 다양한 멀티미디어 응용 서비스들은 MPEG-2, MPEG-4, H.263 등과 같은 동영상 압축 표준 기술을 기반으로 등장, 발전되어 왔다. 그러나 오늘날 새롭게 출현하는 다양한 응용 서비스들은 보다 더 우수한 성능의 압축 기술을 필요로 하고 있다. 이러한 요구에 부응하기 위하여 H.264/AVC라 명명된 새로운 동영상 표준안이 제정되었다[1].

기존 압축 표준에 비하여 H.264/AVC는 강화된 움직임 예측 및 보상 방식, 4x4 블록 단위의 정수 변환, 새로운 엔트로피 부호화 방식, 블록 제거 필터등의 특징적인 기술들을 지니고 있다. 이 같은 특징들로 인하여 H.264/AVC에서는 MPEG-4에 비해 2배정도 압축 성능이 향상되었다[2]. 그러나 H.264/AVC는 세부 과정의 복잡성으로 실제로 구현이 용이하지 않은 문제를 지닌다. 따라서 표준안의 성공적인 활용을 위해서는 효율적인 구현이 이루어져야 한다[3].

화상회의나 비디오폰을 주 응용 분야로 하여 고안된 H.264/AVC Baseline 프로파일에서는 Exp-Golomb 방식과 CAVLC(Context Adaptive Variable Length Coding) 방식이라는 2가지 종류의 엔트로피 부호화 기법에 의하여 구문 요소들이 압축, 복원된다. DCT 계수를 제외한 모든 구문 요소들은 Exp-

* 부산대학교 컴퓨터및정보통신연구소 (yhmoon5@pufs.ac.kr)
논문번호 : KICS2005-10-413, 접수일자 : 2005년 10월 30일, 최종논문접수일자 : 2006년 2월 28일

Golomb 방식에 의하여 부호화된다. Exp- Golomb 방식은 단순한 산술적 연산들로 정의되어지기 때문에 구현이 매우 용이하다는 특징을 지닌다. 한편 4x4 블록에서의 양자화된 DCT 계수들는 CAVLC 방식으로 압축, 복원된다. CAVLC 방식에서는 양자화된 DCT 계수들을 5가지 구문 요소들로 표현하고 이들을 각각 부호화한다. 그 가운데에서 coeff_token, total_zeros, 그리고 Run_before 구문 요소들은 참조 테이블(Look-up Table)에 저장된 코드워드에 의하여 압축, 복원된다. 따라서 이들 3가지 구문요소들은 기본적으로 메모리 액세스를 발생시킨다, 그런데 일반적으로 메모리 액세스는 시스템 동작 속도의 저하 및 전력(Power)소모 증가를 야기한다. 특히 비디오폰 및 DMB폰과 같은 Mobile 플랫폼 기반의 각종 서비스에 있어서 단말기의 Power 소모는 매우 중요한 요소로 인식된다. 따라서 기존의 CAVLC 방법은 실제 서비스 구현에 심각한 문제가 된다.

본 논문에서는 CAVLC 복호화시에 야기되는 메모리 액세스를 감소시키기 위하여 효과적인 Run_before 복원 방식을 제안한다. CAVLC 방식에 있어서 Run_before는 다수의 테이블 참조에 의하여 복원되어진다. 따라서 블록당 빈번한 메모리 액세스가 발생하므로 coeff_token과 total_zeros에 비하여 심각한 문제를 야기할 수 있다. 이를 해결하기 위하여 본 논문에서는 기존 Run_before 복원 방식을 FSM (Finite State Machine) 기법을 이용하여 분석, 해석한다. 그리하여 복원 테이블이 없는 효과적인 Run_before 복원 방식을 제안한다. 제안 방식에서는 메모리 액세스가 불필요하기 때문에 복원 속도의 저하 및 시스템의 Power 소비가 방지되는 효과를 가져온다.

Ⅱ. 기존 CAVLC 복호화 기법

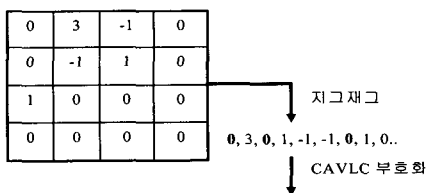
H.264/AVC Baseline 프로파일에서 4x4 오차 블록에 대하여 얻어지는 양자화된 DCT 계수들은 CAVLC 방식으로 압축, 복원된다. 그리고 매크로블록 타입, 움직임 벡터 데이터와 같은 나머지 정보들은 Exp-Golomb 방식으로 부호화된다.

CAVLC 방식에서 양자화된 계수들은 지그재그순으로 재배열된 후 다섯 개의 구문 요소들에 의해 세부적으로 부호화된다. 이들 요소들의 정의는 다음과 같다.

- coeff_token : '0'이 아닌 계수의 수와 크기가 1인 고주파 계수의 수
- sign of T1s : 역 지그재그 순서로 T1 각각의 부호
- level : T1을 제외한 '0'이 아닌 계수의 크기 값
- total_zeros : 지그재그 순서에서 DC와 '0'이 아닌 마지막 고주파 계수 사이에 존재하는 '0'인 계수들의 갯수
- run_before : 역 지그재그 순에서 '0'이 아닌 각각의 계수들 앞에 존재하는 '0'인 계수들의 개수. 즉, '0'인 계수들의 위치 정보

그림 1은 주어진 4x4 블록에 대한 CAVLC 부호화 과정을 나타낸다[4]. 그림 1에서 굵게 표시된 코드워드는 메모리 액세스에 의하여 얻어진다.

한편 CAVLC 복호화에서는 압축된 비트 열로부터 각 구문 요소들의 값을 복원한 후 4x4 블록의 양자화된 DCT 계수들을 구한다. 그림 2는 CAVLC 복호화 과정의 한 예를 보여준다[4]. 그림 2에서 Zero_left는 복원이 이루어지지 않고 남아있는 '0'의 값을 지니는 DCT계수의 갯수를 의미한다. 그림 2에서 T1의 부호화 레벨 요소들은 일반적으로 몇 가지 수식 연산에 의해 복호화된다. 그러나 그림 2에서 coeff_token, total_zero, 그리고 run_before 구문 요소는 복원 테이블을 기반으로 하여 복호화 된다. 이것은 CAVLC 복호화에서 많은 횟수의 메모리 액세스가 요구된다는 사실을 보여준다. 특히, 주어진



구문요소	Value	코드워드
coeff_token	total_coeff=5, T1s=3	**0000100**
sign of T1	+	0
sign of T1	-	1
sign of T1	-	1
level	+1	1
level	+3	0010
total_zeros	3	**111**
run_before	1	**10**
run_before	0	**1**
run_before	0	**1**
run_before	1	**01**

그림 1. CAVLC 부호화 과정의 예

0000100 011 10010 111 101101

비트열	구문요소	Value	복원 상태
0000100	coeff_token	total_coeff=5, T1s=3	x, x, |1|, |1|, |1|
0	sign of T1	+	|1|, |1|, 1
1	sign of T1	-	|1|, -1, 1
1	sign of T1	-	-1, -1, 1
1	level	+1	1, -1,-1,1
0010	level	+3	3, 1, -1, -1, 1
111	total_zeros	3 (zero_left=3)	0,0,0 | 3, 1, -1, -1, 1
10	run_before	1 (zero_left=2)	0,0 | 3, 1, -1, -1, 0, 1
1	run_before	0 (zero_left=2)	0,0 | 3, 1, -1, -1, 0, 1
1	run_before	0 (zero_left=2)	0,0 | 3, 1, -1, -1, 0, 1
01	run_before	1 (zero_left=1)	0 | 3, 0, 1, -1, -1, 0, 1

그림 2. CAVLC 복호화 과정의 예.

오차 블록에 대하여 run_before는 다수의 메모리 액세스에 의하여 복호화가 수행됨을 그림 2로부터 쉽게 알 수 있다. 따라서 복원에 있어서 run_before에 대한 메모리 액세스 감소에 관한 연구가 필요하다.

Ⅲ. 제안하는 run-before 복호화 알고리즘

기존의 CAVLC 복호화기에서 run_before는 표 1의 복원 테이블을 기초로 재귀적 과정(recursive processing)에 의하여 복호화된다. 표 1의 7개 열들 중에서 초기 시작 열은 복원된 total_zeros에 의하여 설정된 초기 zero_left 값에 따라 선택된다. 그림 2의 경우 total_zeros가 3이므로 초기 zero_left 값은 3이 되며 시작 열은 zero_left=3인 열이 된다. 그리고 run_before 값은 입력 비트와 선택된 해당 열의 코드워드의 비교를 통해 결정된다. 즉, 입력 비트와 코드워드가 동일할 때 표 1에서 대응하는 열의 값이 run_before로 결정된다. 한편, zero_left값은 초기의 zero_left 값과 복원된 run_before 값의 차이로 갱신된다. 그리고 부호화를 위하여 대응하는 열이 표 1에서 선택된다. run_before 복호화는 갱신된 zero_left가 '0'이 되거나 '0'이 아닌 계수가 1개 남을 때 까지 반복된다. 이와 같이 재귀적으로 수행되는 기존 복호화 방식은 7개의 상태(state)로 구성된 FSM(finite state machine)으로 구현이 가능함을 쉽게 알 수 있다.

그런데 표 1의 각 열에서 굵은 글씨로 기울어지게 표현된 코드워드들의 경우 그 구조가 비교적 규칙적임을 알 수 있다. 이러한 특성은 복원 테이블의 참조 없이 간단하게 run_before를 복호할 수 있게

표 1. Run-before 복호화를 위한 VLCT

run_before	zero_left						
	1	2	3	4	5	6	7이상
0	1	1	***11***	***11***	***11***	***11***	***111***
1	0	***01***	***10***	***10***	***10***	000	***110***
2		***00***	***01***	***01***	***011***	001	***101***
3			***00***	***001***	***010***	011	***100***
4				***000***	***001***	010	***011***
5					***000***	101	***010***
6						110	001
7							0001
8							00001
9							000001
10							0000001
11							00000001
12							000000001
13							0000000001
14							00000000001

표 2. Run-before 복호화에서의 상태(state) 정의

state	condition
S1	1<=zero_left<=2
S2	3<=zero_left<=5
S3	zero_left=6
S4	zero_left=>7

한다. 예를 들어 zero_left가 5, 입력 비트가 '10'이라고 가정하자. 그러면 run_before는 $3\text{-}10_{(2)}=1$로 계산되며 동시에 zero_left값은 4(=5-1)로 갱신된다. 또한 이러한 코드워드의 구조에 따라 표 1의 모든 열들은 네 개의 그룹으로 분류될 수 있다. 이러한 사실은 run_before 복호화가 수식 연산을 기반으로 한 4개의 상태를 가진 FSM으로 구현될 수 있다는 아이디어를 제공한다. 표 2는 제안하는 run_before 복호화 방식에서 FSM의 상태 정의이다.

제안하는 복호화 방식에 있어서 각각의 상태에서는 run_before 계산, zero_left 갱신, 부호화 종료 검사, 그리고 Next 상태 결정의 4단계를 순차적으로 수행한다.

3.1 run_before 계산

코드 워드의 규칙성에 따라 run_before의 값은 수식 연산에 의해 계산된다. 각 상태에 대한 계산 방식은 다음과 같다.

-S4 상태 : 3 비트 입력[2: 0]에 대해,

$$run\_before = \begin{cases} 7 - I[2:0] & for\ I[2:0] > 0 \\ 4 + m & otherwize \end{cases} \quad (1)$$

-S3 상태 : 3 비트 입력[2: 0]에 대해,

$$run\_before = \begin{cases} I[2:0]+1 & for\ I[2:0] < 2 \\ zero\_left - I[2:0]//2 & for\ I[2:0] \geq 6 \\ I[2:0]+2 & for\ I[2:0] = 2\ or\ 4 \\ I[2:0] & otherwise \end{cases} \quad (2)$$

-S2 상태 : 3 비트 입력[2: 0]에 대해,

$$run\_before = \begin{cases} 3 - I[2:0] & for\ zero\_left \leq 3 + I[2:0]//2 \\ zero\_left - I[2:0] & otherwize \end{cases} \quad (3)$$

-S1 상태 : 2 비트 입력[1: 0]에 대해,

$$run\_before = \begin{cases} (2 - I[1:0]) \cdot (1 - I[1]) & for\ zero\_left = 2 \\ 1 - I[1] & otherwize \end{cases} \quad (4)$$

식 (1)에서 m은 연속되는 비트 '0' 의 총 수를 나타낸다. 한편 다음의 3 단계는 모든 상태에서 공통으로 적용된다.

3.2 zero_left 갱신

zero_left는 1단계에서 얻어진 run_before의 값으로부터 식 (5)와 같이 갱신된다.

$$zero\_left = zero\_left - run\_before \quad (5)$$

3.3 부호화 종료 검사

각 상태에 있어서 갱신된 zero_left가 0 이거나 0이 아닌 계수가 1개 남아있는 경우에 복호화 과정이 종료된다.

3.4 Next 상태 결정

갱신된 zero_left와 표 2의 상태 정의에 따라 다음 상태가 결정된다.

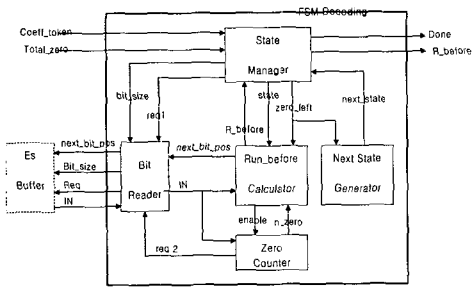


그림 3. 제안하는 run-before 복호화 방식에 대한 시스템 블록도

한편 그림 3은 제안하는 부호화 방식을 토대로 하여 시스템 구조를 보여준다.

그림 3에서 bit_size와 R_before는 각 state에서 읽어들일 입력 비트 수와 최종적으로 구해진 run_before값을 나타낸다. 그리고 Zero counter는 최초의 비트 '1'이전에 발생한 비트 '0'의 개수를 구하는 모듈이다.

Ⅳ. 모의 실험

본 논문에서는 QCIF(176x144) 테스트 영상을 사용하여 기존 복원 방식과 제안 방식간의 성능을 비교, 평가하였다. 이때 탐색 범위는 16으로 설정하였고 5장의 참조 프레임을 사용하였다. 제안 방식은 다양한 QP에 대해서 H.264/AVC 참조 software[5]와 비교하였다. 표 3은 기존 run_before 복호화 방식에서 사용되는 메모리 액세스 횟수이다.

표 3. 기존 run-before 복호화에 요구되는 메모리 액세스 수

QP \ Seq.	Foreman (10fps)	News (10fps)	Container (10fps)	Silent (10fps)
24	387418	243038	166392	185754
28	160034	131232	71258	85420
32	60244	62304	32508	33784
36	19866	26270	14528	10982
40	6346	10238	6070	3440

표 3의 메모리 액세스 횟수는 제안 방식에서 완전히 제거된다. 왜냐하면 복원 테이블의 참조없이 3장에서 서술한 수식들에 의하여 run_before가 복원되기 때문이다. 따라서 제안 방식에서는 복원 테이블을 필요로 하지 않는다. 또한 이때 복원 영상의 화질이 기존 방식에서의 화질과 동일함을 확인하였다. 이상의 사실로부터 제안하는 부호화 방식이 H.264/AVC의 복호화에 요구되는 메모리 액세스를 효과적으로 감소시킬 수 있음을 알 수 있다.

Ⅴ. 결 론

본 논문에서는 CAVLC의 효율적인 복원을 위하여 FSM 기법을 토대로 기존 run_before 복호화 방식을 분석, 해석하였다. 그리고 복원 테이블이 필요없는 새로운 run_before 복호화 방식을 제안하였다. 제안 방식에서는 메모리 액세스가 완전히 제거되기 때문에 시스템의 Power 소모를 예방하고 메모리 액

세스에 따른 속도 저하가 방지된다. 따라서 제안 방식은 DMB나 비디오 폰과 같은 Mobile 플랫폼 환경에 보다 효과적이다.

참 고 문 헌

[1] Joint Video Team of ITU-T and ISO/IEC JTC 1. "Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264 ISO/IEC 14496-10 AVC)", *Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG. doc. JVT-G050.* Mar. 2003.

[2] T. Wiegand, G.J. Sullivan, G. Bjontegard, and A. Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 13, no. 7 pp.560-576, July 2003.

[3] M. Horowitz, A. Joch, F. Kossentini, and A. Hallapuro, "H.264/AVC baseline profile decoder complexity analysis," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 13 no.7 pp.704-716, July 2003.

[4] Lain E.G. Richardson, *H.264 and MPEG-4 Video Compression - video coding for next generation multimedia*, John Wiley & Sons, 2003, pp.187-207.

[5] K. Suhring, Ed., (2002) JM 6.1c Reference Software. Available: http://bs. hhi.de/~suehring/tml/download/jm6.1c.zip

문 용 호 (Yong-ho Moon) 정회원

한국통신학회 논문지 제 29권, 제 1C호 참조