확률분포기반 고속 가변장 복호화 방법

*김은석, 채병조, 오승준
*(주)엠바이엔, 광운대학교 전자공학부
전화 : 031-710-8190, 핸드폰: 018-216-5102

A New Fast Variable Length Decoding Method Based on the Probabilistic Distribution of Symbols in a VLC Table

*Eun-Suk Kim, Byung-Jo Chae, and Seoung-Jun Oh
*MbyN Inc., and School of Electronics Engineering, KwangWoon University
E-mail: sjohu.ac.kr@media.ac.kr

Abstract

Variable length coding (VLC) has been used in many well known standard video coding algorithms such as MPEG and H.26x. However, VLC can not be processed parallelly because of its sequentiality. This sequentiality is a big barrier for implementing a real-time software video codec since parallel schemes can not be applied. In this paper, we propose a new fast VLD (Variable Length Decoding) method based on the probabilistic distribution of symbols in VLC tables used in MPEG as well as H.263 standard codecs. Even though MPEG suggests the table partitioning method, they do not show theoretically why the number of partitioned tables is two or three. We suggest the method for deciding the number of partitioned tables. Applying our scheme to several well-known MPEG-2 test sequences, we can reduce the computational time up to about 10% without any sacrificing video quality.

I. 서론

개인용 컴퓨터 (PC) 성능이 급속하게 향상됨에 따라 고속 연산이 요구되던 멀티미디어 서비스 응용물을 소프트웨어로 처리하려는 움직임이 활발히 진행되고 있다. 특히 최근에는 프로세서 기술의 발달로 MPEG 스트림 복호화를 PC 자체 CPU를 이용하여 소프트웨어로 처리하는 움직임이 활발히 진행되고 있다. PC 개발업자들은 소프트웨어 DVD 재생기를 PC 응용물로 제공하여 고화질의 MPEG-2 스트림을 일반 사용자들이 PC 화면을 통하여 볼 수 있도록 하고 있다. (주)인텔에서 생산하는 펜티움(Pentium) 프로세서는 비디오 데이터 압축 및 복원을 위하여 필요한 처리를 병렬로 할 수 있도록 하는 MMX 기술을 개발하여 비디오 데이터를 이용한 응용물들을 제작할 때 활용토록 하고 있다. MMX는 병렬처리 기법 중에서 SIMD(Singlr Instruction Multiple Data) 방법을 제공하는 것으로 최대 네 쌍의 데이터에 대하여 동일한 연산을 동시에 수행할 수 있도록 하므로 프로그램 개발자가 효율적으로 프로그램을 작성하면 연산 속도를 급증시킬 수 있다.

그러나 MPEG이나 H.26x 등에서 규정한 비쥬얼 데이터 처리 과정을 전부 MMX 코딩 방식을 이용하여 프로그램 할 수 있는 것은 아니다. 특히 가변장 코딩 부분은 처리 과정이 순차적일 수밖에 없으므로 MMX 기법으로 연산 속도를 급증시킬 수 없다. 가변장 코딩은 상기한 표준 기술에서 핵심적이고 가장 빈번하게 수행되는 과정이기 때문에 이를 해결할 방법이 절실히 요구되어 왔다. 본 고에서는 MPEG이나 H.26x와 같이 VLD 처리를 요구하는 시스템에서 사용할 수 있는 최적의 VLD 처리 방법을 제시한다. VLC 표의 특성을

분석하여 표를 적절하게 분할하는 이론을 정립하고, 이 이론에 기반한 분할 방법을 설계한다. 설계한 방법을 소프트웨어 DVD 재생기에 탑재하여 대표적인 실재 시퀀스들에 적용함으로써 그 성능을 입증한다.

II. VLC 표 분할 방법

VLD 과정은 연속적인 시퀀스에서 코드 알파벳(Code Alphabet) C 내에 존재하는 코드를 찾아서 이를 소스로 바꾸는 과정이다. 복호화기에서 VLD 소스 입력 스트림에 대한 확률분포 정보가 없기 때문에 코드 알파벳의 정보를 보고 유추해야 한다. 데이터 시퀀스에서 우리가 원하는 코드를 생성하기 위해서는 비교 과정이 필요하다. 이러한 복호화의 비교함수를 수학적으로 모델링 하려면 비교함수를 정의하여야 한다. 이 비교함수는 전체 모델링 과정에서 중요하게 사용된다.

먼저, 각 비트를 비교하는 함수 $g(c_i^l, c_j^l)$를 정의한다. c_i^l는 원소 코드 c_i의 l+1번째 원소이다. 함수 $g(c_i^l, c_j^l)$는 Exclusive OR 논리함수와 유사하지만, 출력 값으로 세 가지 값 즉, 0, 1, d(don't care) 중에서 한 값을 취한다는 점에서 상이하다. 그러나 함수 $g(c_i^l, c_j^l)$는 원소 코드 와 에서 각 비트를 비교하는 것이므로 원소 코드 자체가 동일한 가를 판단할 수 없기 때문에 두 원소 코드의 동일성을 판별하기 위한 함수 $f_p^q(c_i, c_j)$를 정의한다. 함수 $f_p^q(c_i, c_j)$는 코드 워드 c_i와 c_j 간의 관계만을 나타내므로 모델링을 위해 코드 워드 c_i와 코드 알파벳 C와의 관계를 표현하는 함수 $F_p^q(c_i)_C$를 정의한다. VLD 복호화 과정은 비트열에서 다른 것과 구분되는 코드 워드를 찾는 과정이기 때문에 이러한 함수를 이용하여 구분 가능한 코드를 정의하고, 이 정의를 기반으로 고속 알고리즘을 제안한다. 코드 워드 전체를 한번에 비교할 수 있지만 가변장 코드에서는 각 심볼에 대하여 서로 다른 길이의 코드 워드를 갖게 되므로 모든 코드 워드를 같은 크기로 취급하여 비교하게 되면 가변장 코드 표의 길이가 길어져서 큰 메모리 영역이 필요하다. 그리고 이로 인하여 VLD 하는 시간도 길어지게 된다. 그러므로 가변장 코드 워드를 저장하는 LUT(Look-Up Table)를 분할하여 검사하는 효과적인 방법을 제안한다.

코드를 이 분할하였을 때 크기가 p와 q로 분할된 코드 워드들에 대한 구분 가능을 검사하면, 순서쌍 (p,q)에 대하여 C 내의 원소들은 다음과 같은 두 종류의 부분집합으로 나뉘어진다.

$$D_{C,(p,q)} = \{ c_i \mid F(c_{i,0}|c_{i,1})_C = 0\}\text{와}$$

$$N^i_{C,(p,q)} = \{ c_i \mid F(c_{i,0}|c_{i,1})_{C-D} = 0 \text{ and } F(c_{i,0}|c_{i,1})_{N_i} \neq 0\}$$

$$C = D_{C,(p,q)} \cup N^0_{C,(p,q)} \cup \ldots \cup N^r_{C,(p,q)}$$

윗 식에서 연산자 "|"는 지면의 제약 때문에 원 논문에서 설명할 예정이다. D와 N에 대한 다이어그램을 그림 1에 보였다. 집합 D와 집합 N^i로 나누는 알고리즘은 다음과 같다.

1) 초기 상태에는 모두 집합 D라 하고, 집합 N^i는 없다고 가정한다.
2) 코드들을 하나 정해서 다른 모든 코드와 구분 가능 관계를 살핀다.
3) 구분 가능하지 않은 코드가 존재하면 집합 D에서 제외하고 집합 N^i로 구분한다.
4) 다른 집합 N^k (단 $k \neq i$)에 포함된 코드와 구분가능 관계를 살피고 구분 가능하지 않은 코드가 존재한다면 두 개의 집합을 합한다. 만약에 구분 가능하지 않은 코드가 없다면 N^i 를 새로운 집합으로 규정한다.
5) 2)-4)의 과정을 모든 코드가 조사될 때까지 반복한다.

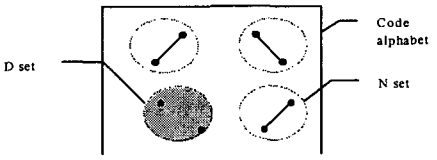


그림 1. 연결 다이어그램

그림 2는 분할 과정의 구조인 트리 형태로 LUT를 구분하는 방법을 보여준다. VLD 과정을 수행하기 위하여 요구되는 시간을 확률적으로 분석하기 위해서는 비교하는 단위 과정에 대한 시간 분석이 필요하다. 여기서 단위 과정이란 일정한 비트를 비교하고 LUT 내에서 찾는 과정을 말하며, 그림 3과 같이 표시할 수 있다. 그림 3처럼 단위 과정은 Truncation과정, Show bits 과정, 검색 과정, Branch jumping과정, Flush bits 과정 등 다섯 과정으로 나눌 수 있다. 특별한 단위 과정으로는 초기 단위와 결정 단위가 있는데, 상기한 알고리즘에서 초기 단위에서는 1)번 과정이 생략되고, D 밖에 존재하지 않는 결정 단위에는 4)번 과정이 생략된다. 단위 과정을 수행하는 시간은 다음 식과 같다. T_k는 k 과정을 수행하기 위하여 요구되는 시간이다.

$$T_{UP(C)} = T_{show(p)} + T_{search} + P(D) \times T_{flush} + \sum_i P(N_i) \times (T_{flush(l_{C_i} - l_{N_i})} - T_{UP(N_i)})$$

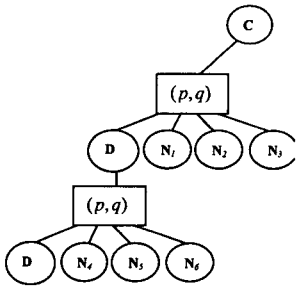


그림 2. 트리 형태의 LUT 구분법

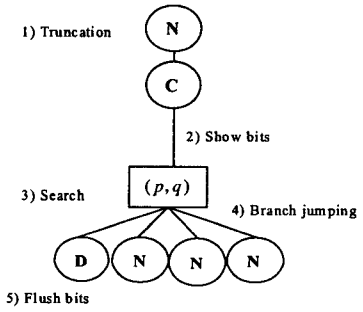


그림 3. 단위 과정

먼저 원하는 비트를 받아들이는 루틴에 대한 수학적인 분석한다. 그림 4와 같이 데이터는 저장 매체에 존재한다. 이 데이터를 메모리 상에 존재하는 L2 버퍼로 옮기고, 비트 연산을 위해 다시 L1 버퍼로 옮긴다. L1 버퍼의 크기 l_{L1}은 비트 연산이 가능한 최대 단위이며, MMX와 같이 병렬처리가 가능한 프로세서에서는 64비트이며 MMX 명령어를 지원하지 않는 프로세서에서는 32비트이다.

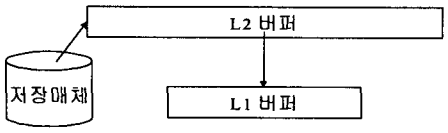


그림 4. 비트 입력 루틴 블록 다이어그램

L1 버퍼 내에는 사용된 비트와 사용되지 않은 비트가 존재한다. 사용되지 않은 비트량을 l_a로 표시하고 얻고자 하는 비트량을 l_{show}라 하면 l_{show}가 l_a보다 작거나 같은 경우는 단순히 앞뒤로 비트 이동를 함으로써 쉽게 구할 수 있다. 그렇지 않은 경우는 L1내에서 사용되지 않은 영역을 뽑아낸 후에 다시 L2에서 L1로 데이터를 이동하여 다시 비트 연산을 수행한 다음 두 데이터를 합치는 과정이 필요하다. 리틀 인디언(Little Endian) 구조를 갖는 프로세서에서는 바이트 순서를 치환하는 과정이 추가적으로 필요하다. 즉, L1내에 필요한 비트가 모두 존재하는 가에 따라 데이터를 가지고 오는 시간이 매우 차이가 난다. 전자의 경우 걸리는 시간을 T_1이라고 하고 후자의 경우를 T_2라고 하면 T_1과 T_2 간에는 $T_1 \ll T_2$이 성립한다. 원하는 비트를 얻어오는 평균 시간은 다음 식과 같다.

$$T_{show} = P(l_a \geq l_{show}) \cdot T_1 + P(l_a < l_{show}) \cdot T_2$$

일반적으로 $P(l_a)$는 l_a에 $P(l_a = 0)$의 l_a가 작은 경우가 확률이 높고 다른 분포를 볼 수 있다. 예를 들어, 바이트 정렬이 된 경우에는 균일 분포(Uniform Distribution)로 가정할 수 있다. 코드 집합에서 l_{show} 만큼을 살펴보고 LUT내에서 해당 코드를 찾아 낼 수 있는 확률은 다음 식과 같다.

$$P(C_{show} \in D) = \sum_{l_{C_i} < l_{show}} P(C_i) = \sum_{l_{C_i} < l_{show}} \frac{1}{l_{C_i}}$$

III. 실험 결과

제안한 방법을 MPEG-2 스트림에 적용하여 타당성을 보였다. MPEG-2에서 VLC시에 사용하는 비트의 최대 크기는 17비트이다. 17비트를 이용하여 LUT를 만들 때에는 약 128KB의 데이터가 필요하다. 128KB가 큰 양이 아니더라도 이 크기의 데이터를 자주 처리하면 캐쉬의 히트율(Hit Ratio)이 떨어지게 되어 검색 시간을 증가시킨다. 그리고 17비트를 검토할 때 사용할 비트의 양이 많아지므로 더 많은 시간을 소모할 확률이 매우 높아지게 된다. 따라서 캐쉬의 히트율도 높이고 레지스터 내의 유효한 비트 내에서 원하는 데이터를 얻어낼 확률을 높이기 위하여 LUT를 제안한 방법으로 나누었다. 사용한 스트림에 대해서는 2개로 표를 나누는 것이 최적으로 판정되었다. 분할된 LUT 내에서 코드를 구별할 수 있어야 하기 때문에 특정한 위치에서만 나뉘어지게 되어 LUT가 겹친다. 첫 번째 LUT는 9비트를 사용하였고 두 번째 LUT는 11비트를 사용하였기 때문에 3비트가 겹친다. LUT-1은 9비트이므로 512B(Byte) 크기이고 LUT-2는 11비트를 사용하기 때문에 2kB 크기이다. 따라서 두 LUT를 합치더라도 2.5kB이므로 펜티엄 프로세서의 페이지 크기인 4kB보다 적어서 캐쉬 히트율을 높일 수 있다. 그리고 LUT-1만을 보면 17비트를 사용할 때보다 사용될 비트가 레지스터에 있을 확률이 매우 높아지게 된다. 만약에 유효한 비트의 양이 균일 분포를 가진다면 확률은 2배에 가깝게 되어 시간을 단축시키는데 큰 역할을

한다. 그림 5는 H.263에서 사용하는 VLC 코드의 이론적인 확률 분포를 보여준다.

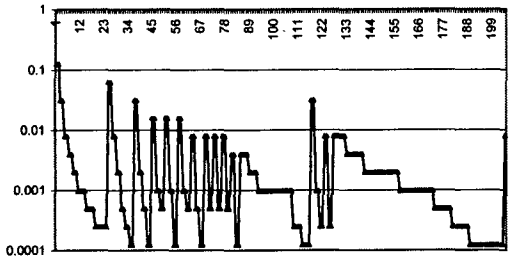


그림 5. *H.263에서 사용하는 VLC코드의 이론적인 확률 분포*

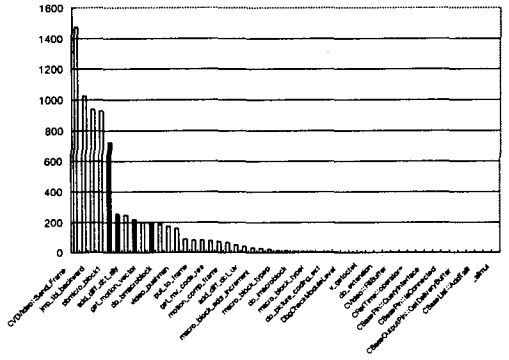
제안한 방법의 성능을 평가하기 위하여 인텔의 VTune을 사용하여 20초 정도를 복호화한 후에 각 모듈의 수행 시간을 비교하여 성능을 분석하였다. 그러나 실행될 때마다 약간씩 차이가 나므로 이를 총 실행시간(D), IDCT의 시간(E)과의 비로써 측정하였다. 모든 형태의 복호화 방법이 모두 포함되어 있고 coefficient zero와 coefficient one을 모두 사용한 Vts4_1.vob 스트림을 사용하여 각각의 방법에 대하여 세 차례 실시한 후 비율을 평균 내었다. 표 1에 각각의 방법의 성능을 분석하여 정리하였다. 총 실행시간 면에서 비교하였을 때 약 10%정도 성능이 향상되었다. 그림 8에 결과를 정리하였다.

표 1. VTune을 이용한 성능분석

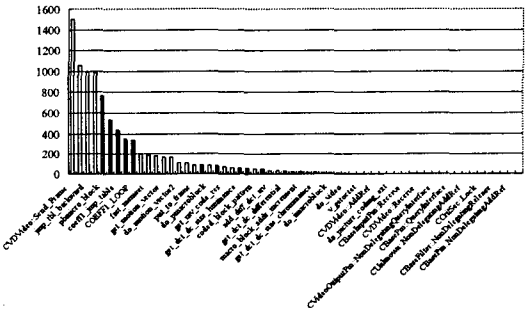
구분	기존 방법	제안한 방법
총실행시간과의 비교	0.3	0.18
IDCT와의 비교	2.74	1.5

IV. 결론

가변장 코딩은 순차적인 특성 때문에 병렬처리를 통하여 성능을 향상시킬 수 없다. 그러나 가변장 코딩은 최근에 멀티미디어 데이터 처리를 위한 국제 표준에서 가장 보편적으로 사용되는 기술이다. MPEG이나 H.263과 같은 국제 표준 기술에서 핵심적이고 가장 빈번하게 수행되는 과정이기 때문에 이를 해결할 방법이 절실히 요구되어왔다. MPEG 표준화 단체나 H.26x 표준화 단체 등에서 제공하는 TM(Test Model)이나 VM(Verification Model)에서는 VLD 연산을 방법으로 VLC 표를 분할하는 방식을 제공하고 있었지만 모든 비디오 시퀀스에 일률적으로 적용하기 때문에 입력 데이터 시퀀스에 최적인 분할 방법을 제공하지 못하였다. 본 고에서는 VLC 표의 특성을 분석하여 입력 데이터 시퀀스 특성에 맞추어서 VLC표를 적절하게 분할하는 이론적인 근거를 제시하였고, 이에 따라 VLC표를 몇 개로 분할 할 것인가를 결정하는 방법을 제안하였다. 제안한 방법을 MPEG-2 스트림을 재생하는 소프트웨어 DVD 재생기에 탑재하여 대표적인 MPEG-2 시험 데이터 시퀀스들에 적용하였을 때 약 10% 정도의 성능 향상을 얻을 수 있었다



(a) 제안한 고속 VLD



(b) 기존 VLD

그림 8. VLC관련 루틴의 시간 소모량

참고문헌

[1] J.L. Mitchell *et al.*, *MPEG Video Compression Standard*, Chapman & Hall Inc., N.Y., 1996

[2] ISO/IEC JTC1/SC29/WG11, *Generic coding of moving pictures and associated audio, Part 2:Video, IS 11172-2*, Aug. 1993

[3] ISO/IEC JTC1/SC29/WG11, *Generic coding of moving pictures and associated audio, Part 2:Video, IS 13818-2*, March 1995

[4] ITU-T, *Draft ITU-T Recommendation H.263*, 1998