효율적인 테이블 메모리를 갖는 가역 가변길이 부호

임 선 웅(任 善 雄), 배 황 식(裵 皇 植), 정 정 화(鄭 正 和)
한양대학교 전자공학과 CAD 및 통신회로 연구실
전화 : (02) 2290-0558 / 팩스 : (02) 2299-2129

A reversible variable length code with an efficient table memory

Sun-woong Im, Hwang-sik Bae, Jong-wha Chong
CAD & C.C. Lab., Dept. of Electronic Eng., Hanyang Univ.
17 Haengdang-Dong, Seongdong-Gu, Seoul, Korea

E-mail : swimmer@hymail.hanyang.ac.kr

Abstract

A RVLC(Reversible Variable Length Code) with an efficient table memory is proposed in this paper.

In the conventional decoding methods, the weight of symbols and code values are used for the decoding table. These methods can be applied for Huffman decoding. In VLC decoding, many studies have been done for memory efffciency and decoding speed.

We propose an improved table construction method for general VLC and RVLC decoding, which uses the transition number of bits within a symbol with an enhanced weight decomposition.

In this method, the table for RVLC decoding can be implemented with a smaller memory

I. Introduction

최근 기술의 발전으로 인한 이동 통신의 급속한 성장은 영상, 음성등의 멀티미디어 데이터의 신뢰성 있는 전송을 요구하게 되었다.

비디오 압축기술은 대역폭이 제한된 채널에서 영상 정보들을 보내는 대역폭을 줄일 수 있게 해준다. 이러한 기술중의 하나로 가변길이 부호화(Variable Length Coding)가 채택되는데, 가변길이 부호화의 경우 대부분 허프만 부호화(Huffman Coding)가 쓰인다. 그리고 최근 MPEG-4등에서 채택되고 있는 새로운 가변길이 부호가 있는데, 바로 가역 가변길이 부호(Reversible Variable Length Code)로서, 기존의 가변길이 부호에 가역성(reversibility)를 추가한 것이다. 가역성을 추가하였기 때문에 기존의 가변 길이 복호에서 오류가 발생했을 때 버려야 했던 데이터들 중 상당 부분을 복원해낼 수 있는 장점이 있다.

이러한 가변길이 부호들을 복호해내기 위해서는 룩업 테이블을 갖고 있어야 하는데, 이 논문에서는 이러한 테이블 메모리(table memory)를 효율적으로 구성하여 보다 적은 메모리로 구현하는 방법을 제안한다.

II. Prerequisites

Huffman coding으로 알려진 가변길이 부호화는 JPEG, MPEG, 기타 이미지 데이터 표준에서 널리 사용된 데이터 압축 방법중의 하나이다. 가변길이 부호화의 기본원리는 이산여현변환(DCT, Discrete Cosine Transform)이나 양자화(quantization)를 수행한 후에 통계적인 잉여항(statistical redundancy)를 제거하는 것이다.

이러한 가변길이 부호들을 복호할 때의 문제점은 복호과정중 오류(error)가 발생했을 때 그림.1에서처럼 동기(synchronization)를 잃게 되고 결과적으로 다음 동기까지의 모든 데이터를 복호할 수 없게 된다. 이러한 문제점으로 인하여 최근 가역 가변길이 부호(RVLC, reversible variable length code)에 대한 연구가 활성화되고 있다.

MPEG-4의 Error resilience scheme 중의 하나로도 채택된 가역 가변길이 부호는 기존의 가변길이 부호에서 불가능했던 역방향 복호(backward decoding)가 가능하다. 그림.2에서 처럼 역방향 부호화를 통해서 기존

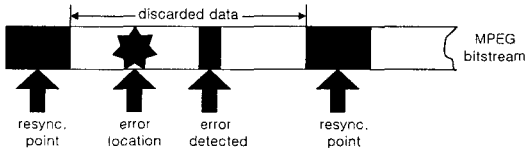

그림 1. 일반적인 가변길이 부호(VLC)의 오류처리

의 가변 길이 부호에서 잃을 수 밖에 없던 데이터의 상당 부분을 복구할 수 있다. 본 논문에서는 이러한 가역 가변길이 부호의 복호시 사용되는 테이블 메모리를 효율적으로 구성하는 방법을 제안한다.

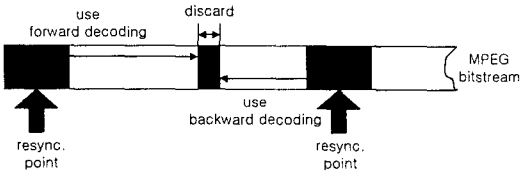

그림 2. 가역 가변길이 부호(RVLC)의 오류처리

III. Two schemes of RVLC

본 논문에서는 RVLC 에 대한 여러 연구중에서 코드의 대칭성을 기준으로 분리한 대칭 가역 가변길이 부호(symmetric RVLC)와 비대칭 가역 가변길이 부호(asymmetric RVLC)에 대하여 다룬다[1].

부호들을 다루기에 앞서, 가역 가변길이 부호가 되기 위해서는 두가지 조건이 만족되어야 하는데, 이는 바로 접두부호 조건(prefix condition)과 접미부호 조건(suffix condition)이다. 접두부호 조건은 각각의 부호가 자신보다 더 긴 부호의 앞부분과 일치하면 안 되는 것을 의미하는데, 이 조건은 기존의 비가역 가변길이 부호에도 적용된다. 접미부호 조건이 있는데, 이것은 각각의 부호가 자신보다 더 긴 부호의 뒷부분과 일치하면 안 되는 것을 의미하는데, 이 조건은 가역 가변길이 부호(RVLC)의 역방향 복호(backward decoding)시에 올바른 부호로의 해석을 보장한다.

III-1. 대칭 가역 가변길이 부호 (symmetric RVLC)

대칭 가역 가변길이 부호는 그 대칭성으로 인해, 메모리 사용량과 부호의 간결함의 측면에서 좋은 성능을 나타낸다. 대칭 가역 가변길이 부호를 만드는 방법은 다음과 같다. 기존의 비가역 가변길이 부호를 하나하나 따라가면서 위에서 언급한 접두부호 조건과 접미부호 조건을 만족하지 않는 부호는 대칭 부호 중 다른 것으로 바꿔주면 된다. 이를 그림.3과 그림.4를 통해 설명하면 다음과 같다.

그림.3의 (a)는 5개의 부호로 구성된 간단한 비가역 가변길이 부호의 한 예이다. 부호 A는 '00'인데, 부호가 대칭이며 접두부호 조건과 접미부호 조건에 위배되지 않는다. 부호 B는 '01'로 구성되어 있어서 대칭성을 만족시키지 못하므로, 다른 부호로 바꿔줘야 하는데,

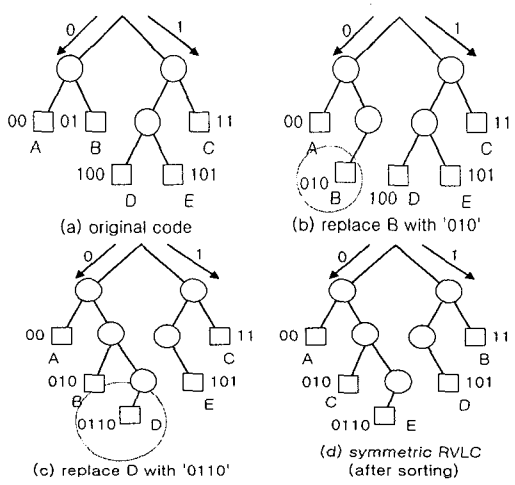

그림 3. 대칭 가역 가변길이 부호 생성 예

그림.4에서 보면 같은 레벨의 '11'부호는 이미 C로 할당되어 있다. 다음 레벨로 넘어가서 '000'부호가 있는데, 이 부호를 선택하면 이미 부호 A로 선택된 '00'때문에 접두부호 조건과 접미부호 조건에 위배된다. 따라서, 그 다음 부호인 '010'을 B로 할당한다. 부호 C는 대칭이며 접두부호 조건, 접미부호 조건을 만족시킨다. 부호 D는 대칭이 아니다. 따라서, 대칭인 부호로 바꿔줘야 하는데, 같은 레벨의 '101'은 이미 부호 E로 할당되어 있고 '111'은 부호 C와 접두부호 조건과 접미부호 조건에 위배되므로 다음 레벨로 넘어간다. 다음 레벨에서 부호'0000'은 부호 A와 접두부호 조건, 접미부호 조건에 위배되므로, 그 다음 대칭 부호인 '0110'을 부호 D로 할당한다. 그리고 마지막 부호 E는 '101'로 대칭이며, 다른 부호와 접두부호 조건, 접미부호 조건에 위배되지 않으므로 이로써 대칭 가역 가변길이 부호로의 전환이 끝났고, 정렬(sorting)을 수행함으로써 최종적인 결과인 그림 3.(d)를 얻는다.

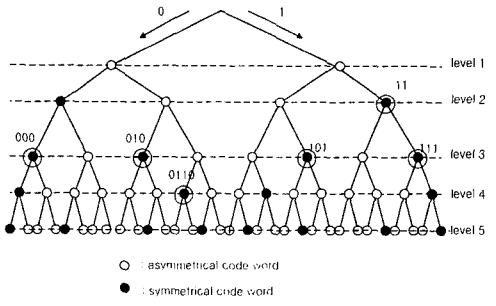

그림 4. 대칭 가역 가변길이 부호 이진 트리

III-2. 비대칭 가역 가변길이 부호 (Asymmetric RVLC)

비대칭 가역 가변길이 부호는 대칭성을 고려하지 않고 부호를 생성하기 때문에 부호 생성시 대칭 가역 가변길이 부호보다 더 좋은 효율을 나타낸다.

그림.5를 보면서 생성과정을 설명해 보면 다음과 같다. 그림.5의 (a)는 4개의 부호로 구성된 기존의 비가역 가변길이 부호의 한 예이다. 그림.5의 (b)처럼 역이진 트리를 만든다. 그 다음 기존의 부호를 따라가면서, 각 부호의 접미부호가 짧은 부호의 접두부호에 나올 경우 새로운 부호를 할당함으로써 비대칭 가역 가변길이 부호를 만들게 된다. 그림.5에 있는 부호의 경우 부호 C를 만들 때 문제가 발생한다. '001'이란 부호를 C로 할당하면, 부호 A가 부호 C의 접두부호와 동일하게 되므로 다른 부호로 할당해주어야 한다. 같은 레벨에 할당할 수 있는 부호가 없을 때는 최소수의 비트(bit)를 추가해줌으로써 새로운 부호를 할당한다. 이 경우 '1001'의 부호를 부호 C로 할당할 수 있다. 부호의 할당이 끝난 다음 대칭 가역 가변길이 부호의 경우와 마찬가지로 정렬을 수행함으로써, 비대칭 가역 가변길이 부호의 생성을 완성한다.

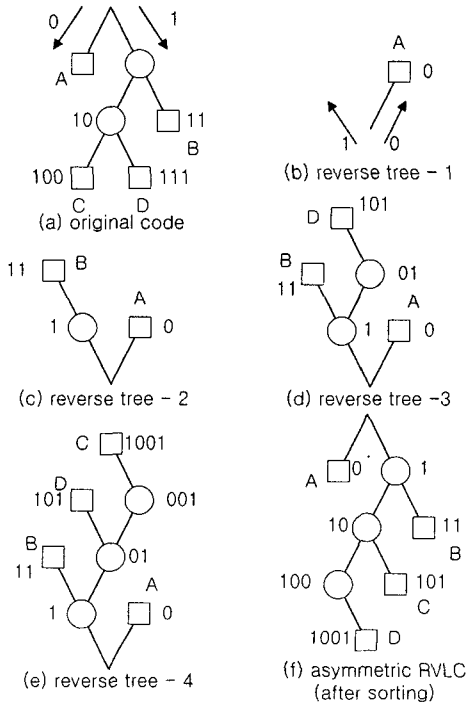

그림 5. 비대칭 가역 가변길이 부호 생성 예

IV. 효율적인 테이블 메모리 사용의 복호 알고리듬

III.에서 완성된 부호들은 테이블 메모리에 저장되어야 한다. 본 논문에서는 가변길이 복호에 쓰이는 각 코드들의 천이 개수(transition number)와 가중치(weight)를 이용하여 효율적으로 테이블 메모리를 구성하는 방법을 제안한다.

효율적으로 테이블 메모리를 구성하는 방법은 다음과 같이 간단하게 설명될 수 있다. 각각의 코드들의 천이 개수와 가중치를 구한다음 각각을 2의 멱승(power)형태로 만든후, 배열(array)에 저장한다. 가중치는 테이블을 구성하는 트리의 가장 큰 레벨의 값에서 자신의 레벨을 뺀 값을 2의 멱승형태로 표현한 값이다. 예를 들어 어떤 가변길이 복호에 사용된 테이블의 가장 긴 코드의 레벨이 7이라고 한다면 레벨이 2인 코드의 가중치는 $2^{(7-5)} = 2$ 가 된다. 그 다음 부호들의 혼동을 피할 수 있도록 값들을 재배열한 후에 세 개의 값들을 하나씩 묶음으로써 최종적으로 우리가 원하는 테이블을 얻는다.

테이블을 압축하기에 앞서 값들을 재배열하는 방법은 다음과 같다.

재배열 과정중 가중치와 천이 개수가 같을 수 있는데, 이런 경우가 발생했을 때에는 서로 구분할 수 있는 특징을 추출하여 천이 개수가 들어갈 자리에 넣어주고 이는 복호(decoding)시 사용된다. 본 논문이 제안하는 방법의 특징은 가장 먼저 나오는 '1' 비트의 위치를 이용하는 것이다. 즉, 같은 가중치와 천이 개수를 가진 부호가 있을 때, 부호의 가장 처음 나오는 '1'비트의 위치를 천이 개수와 함께 더하여 2의 멱승형태의 값으로 저장하게 된다. 이렇게 재구성된 값을 다시 내림차순으로 정리하여 배열내에서의 위치를 바꿔주게 된다. 또한 같은 가중치와 천이 개수를 가지면서 위의 규칙으로 구분할 수 없는 부호들이 발생할 경우에는 구분할 수 있는 부호들을 배열의 앞부분에 위치시키고, 다른 부호들과 구분이 모호한 것은 상대적으로 배열의 뒷부분에 위치시켜서 복호시 부호를 잘못 구분하는 일이 없게 한다.

천이 개수를 분리했을 때 다른 가중치를 가지면서 같은 천이 개수를 가지는 부호가 발생했을 때에는 인근 배열의 부호와 바꿔줌으로써 문제를 해결한다.

제안한 알고리듬을 검증한 RVLC 테이블은 비대칭 가역 가변길이 부호이다[1]. 표.1은 원래 부호들의 가중치와 천이 개수를 갖고 있는 테이블이다. 표.2는 위에서 설명한대로 부호들을 확실히 구분해하기 위해 원래의 테이블에 변형 작업을 수행한 테이블이다. 표.3은 제안된 알고리듬에 의해 최종적으로 압축된 테이블이

symbol	E	T	A	O	R	N	H	I	S	D	L	U	P	F	M	C	W	G	Y	B	V	K	X	J	Q	Z
weight	1024	1024	512	512	512	512	512	256	256	256	256	256	256	256	128	128	128	128	32	32	16	16	8	8	4	1
no. of transition	1	1	0	2	3	1	3	3	2	2	1	2	1	0	2	4	3	2	3	4	4	3	3	6	6	6

표.1 가중치와 천이개수를 나타낸 테이블

symbol	T	E	A	I	O	N	H	R	F	U	D	L	S	P	C	M	G	W	J	B	Y	Q	V	X	Z	K
weight	1024	1024	512	256	512	512	512	512	256	256	256	256	256	256	128	128	128	128	8	32	32	4	16	8	1	16
decision	16	4	1	8	4	2	128	64	1	128	16	8	64	32	16	32	16	8	64	16	8	64	16	8	64	8

표.2 압축을 위해 변형된 테이블

i	0	1	2	3	4	5	6	7	8
weight	2560	1280	1280	768	640	384	72	28	17
decision	21	14	193	152	112	56	88	88	72

표.3 압축된 테이블

다. 제안된 알고리듬은 psedo-code로 다음과 같이 설명된다.

(RVLC memory effcient decoding algorithm)

1. $i \leftarrow 0$

2. if $2^{\log x} = W_i$, *then* $A = B = 2^{\log x}$

   otherwise $A = 2^{\log x}$ and $B = W_i - A$

   2-1. weights type I

   $w_1 = 2^{\log A}$, $w_2 = A - w_1$, $w_3 = B$

   2-2. weights type II

   $w_1 = 2^{\log B}$, $w_2 = B - w_1$, $w_3 = A$

   if weight of code are one of weights of two types, then go to step 3; *otherwise* $i \leftarrow i+1$ and go to step 2.

3. decision values are decomposed into 3 values by method used in step 2.

   3 values values are d_1, d_2, d_3 ($d_1 > d_2 > d_3$).

   trn : transition number of code

   if $2^{trn} = d_1$ or $(\log_2 d_1 - trn)$th bit of code = 1

   ,then code is $3i$th symbol of RVLC.

   else if $2^{trn} = d_2$

   or $(\log_2 d_2 - trn)$th bit of code = 1

   ,*then* code is $(3i+1)$th symbol of RVLC.

   else if $2^{trn} = d_3$

   or $(\log_2 d_3 - trn)$th bit of code = 1

   ,*then* code is $(3i+2)$th symbol of RVLC.

   else if i indicates last array of weight,

   .*then* go to step 4.

   else code is not symbol of VLC or RVLC.

4. *if* $W_i = A + B$ and $2^{trn} = d_1 + d_2 + d_3$

   ,*then* code is $3i$th symbol of RVLC.

   else if $W_i = A$ *or* B

   and ($2^{trn} = d_1 + d_2$ *or* $d_1 + d_3$

   or $(\log_2 (d_1 + d_2) - trn)$th bit of code = 1

   or $(\log_2 (d_1 + d_3) - trn)$th bit of code = 1)

   ,*then* code is $3i$th symbol of RVLC.

   else if ($2^{trn} = d_2$ *or* d_3

   or $(\log_2 d_2 - trn)$th bit of code = 1

   or $(\log_2 d_3 - trn)$th bit of code = 1)

   ,*then* code is $(3i+1)$th symbol of RVLC.

   else code is not symbol of RVLC.

V. 결론

본 논문에서는 가역 가변길이 부호를 효율적으로 테이블 메모리에 저장하는 방법을 제안했으며, 기존의 방법[2]보다 테이블 메모리의 양을 약 33%정도 줄일 수 있다.

또한 본 연구는 보다 효율적인 부호 구분을 위해 계속 진행되고 있다.

References

[1] Y. Takishima, M.Wada, H.Murakami, "Reversible Variable Length Codes," *IEEE trans. Communications*,vol43.NO.2/3/4/,1995, pp.158-162

[2] H.C.Chen, Y.L Wang. Y.F Lan.,"A memory-efficient and fast Huffman decoding algorithm," *Information Processing Letters*, vol 69, 1999, pp. 119-122

[3] Raj Talluri, "Error-Resilient Video coding in the ISO MPEG-4 standard," *IEEE Communications Magazine*, June 1998, pp. 112-119

[4] 이승준, 서기범, 정정화, "A Memory-effcient VLC Decoder Architecture for MPEG-2 Application," 대한전자공학회 추계종합학술대회논문집, 제22권 제2호, pp 360-363, 1999