Turbo code를 위한 효율적인 SOVA의 구현

정회원 이 창 우*

Efficient Implementation of SOVA for Turbo Codes

Chang-Woo Lee* ***Regular member***

요 약

Turbo code를 비롯한 concatenated code를 복호하기 위한 알고리즘 중에서 SOVA(soft output Viterbi algorithm)는 최적의 MAP(maximum a posteriori) 알고리즘이나 준최적인 Max-Log-MAP 알고리즘에 비해서 성능은 다소 떨어지는 반면에 계산량이 가장 적은 알고리즘이다. 본 논문에서는 SOVA 구현을 위한 soft decision 값을 갖는 출력 계산과 trace-back 연산을 모든 competing path에 대해서 동시에 진행함으로써 복호 시간을 줄이는 방법을 제안한다. 또한 전체 블록을 부블럭으로 나누어서 제안하는 알고리즘을 적용하여 memory를 효율적으로 사용하는 기법을 제안한다. 3GPP 표준에서 사용되는 turbo code를 이용한 성능 분석 결과 제안하는 알고리즘이 기존의 SOVA 구현 기법과 동일한 결과를 보이면서 65%~75% 정도의 계산량 만을 필요로 하는 것을 알 수 있었다.

Key Words: turbo code, SOVA, MAP, concatenated code.

ABSTRACT

The SOVA, which produces the soft decision value, can be used as a sub-optimum solution for concatenated codes such as turbo codes, since it is computationally efficient compared with the optimum MAP algorithm. In this paper, we propose an efficient implementation of the SOVA used for decoding turbo codes, by reducing the number of calculations for soft decision values and trace-back operations. In order to utilize the memory efficiently, the whole block of turbo codes is divided into several sub-blocks in the proposed algorithm. It is demonstrated that the proposed algorithm requires less computation than the conventional algorithm, while providing the same overall performance.

I. 서 론

1993년에 Berrou 등이 제안한 turbo code는 Shannon의 이론적 한계치에 근접하는 성능을 보이는 오류 정정 부호이다[1]. Turbo code의 복호를 위해서는 MAP(maximum a posteriori) 알고리즘, Log-MAP 알고리즘, Max-Log-MAP 알고리즘, SOVA(soft output Viterbi algorithm), modified SOVA 등이 사용될 수 있다. 이중에서 MAP 알고리즘이 성능상 최적으로 알려져 있는데 다수의 곱셈과 exponential 계산 등으로 인해서 복호 시간이 오래 걸리는 단점이 있다. 특히 turbo code는 복호를 위해서 다수의 iteration이 필요하므로 보다 효율적인 알고리즘에 대한 연구가 활발히 진행되어 왔다[2-4]. Log 연산의 특성을 이용하여 MAP 알고리즘과 동일한 성능을 보이면서 계산량을 줄인 것이 Log-MAP 알고리즘이고 Log-MAP 알고리즘을 근사화하여 성능상의 손실을 감수하고 계산량을 더욱 줄인 것이 Max-Log-MAP 알고리즘이다. 한편 convolutional code를 위한 MLSE(maximum likelihood sequence estimation)로서 사용되는 Viterbi 알고리즘을 수

* 가톨릭대학교 정보통신전자공학부(changwoo@catholic.ac.kr)

논문번호: 030201-0513, 접수일자: 2003년 5월 14일

※본 연구는 한국과학재단 목적기초연구(No. R05-2001-000-00908-0) 지원으로 수행되었음.

정하여 turbo code와 같은 concatenated code의 복호에 적합하도록 soft decision value를 출력으로 하는 알고리즘이 SOVA이다. SOVA를 사용하면 계산량은 MAP을 기반으로 하는 알고리즘에 비해서 작지만 성능은 다소 떨어진다. 또한 SOVA를 변형하여 복잡도는 높아지지만 성능을 좋게 향상시킨 modified SOVA가 Max-Log-MAP과 동일한 성능을 보인다는 사실이 입증되었다[5]. 만일 turbo code를 대용량의 멀티미디어 데이터 등의 실시간 처리에 적용하기 위해서는 turbo code의 복호에 요구되는 계산량을 줄이는 것이 관건이 되기 때문에 복호 알고리즘으로 계산량이 가장 작은 SOVA를 사용할 수 있다.

본 논문에서는 대용량의 멀티미디어 정보에 turbo code를 적용하기 위해서 기존의 알고리즘 중에서 복호 시간이 가장 작은 SOVA의 성능을 저하시키지 않고 계산량을 더욱 줄이는 효율적인 알고리즘을 제안한다. 기존의 SOVA에서는 surviving path와 합쳐지는 모든 competing path를 독립적으로 trace-back하면서 difference metric과 log-likelihood ratio를 비교하고 갱신하는 과정으로 이루어져 있다. 이러한 과정을 각 stage에서 모든 competing path에 대해서 동시에 수행하여 log-likelihood ratio 갱신과 trace-back 연산의 횟수를 줄이는 기법을 제안한다. 또한 turbo code를 복호하기 위해서는 각 stage의 metric을 저장하기 위한 memory가 필요하다. Turbo code는 일반적으로 블록의 크기가 커질수록 성능이 좋아지는데 이를 구현하기 위한 memory의 크기도 같이 커지는 문제가 있다. Memory를 효율적으로 사용하기 위해서 전체 블록을 일정한 크기의 부블럭으로 나누어서 turbo code를 복호하는 알고리즘이 제안되었다[6]. 본 논문에서는 이러한 기법을 제안하는 SOVA의 구현에 맞도록 변형하여 적용함으로써 memory를 효율적으로 사용할 수 있도록 하였다. 특히 제안하는 알고리즘에서는 trace-back 연산을 부블럭 단위로 수행할 수 있기 때문에 더욱 효율적인 구현이 가능하다. 제안하는 알고리즘을 3GPP 표준의 turbo code[7]에 적용하였을 때 AWGN(additive white Gaussian noise) 채널 환경에서 BER(bit error rate) 성능과 계산량을 기존의 SOVA와 비교하여 제안하는 알고리즘이 동일한 성능을 보이면서 계산량을 크게 감소시키는 것을 확인하였다

Ⅱ. SOVA

Turbo 부호기는 일반적으로 2 개의 RSC(recursive systematic convolutional) 부호기와 인터리버로 구성된다. 인터리버를 이용하여 상관성이 제거된 정보에 대해서 각각의 부호기에서 parity 비트를 생성하여 정보 비트와 함께 전송하고 복호기에서는 두 RSC code에 대한 복호 결과를 이용하여 반복적인 복호를 통해서 성능을 높일 수 있다. 그림 1에 turbo code의 복호기를 도시하였는데 각 RSC code의 복호기는 다른 복호기의 log-likelihood ratio 출력을 사전 확률(APP: a priori probability)로서 branch metric의 계산에 이용한다. 따라서 turbo 복호기를 구성하는 두 개의 RSC code를 위한 복호기는 soft decision value를 출력할 수 있어야 한다.

그런데 convolutional code를 복호하기 위한 MLSE로서 제안된 Viterbi 알고리즘은 hard decision value를 출력하기 때문에 turbo code와 같은 concatenated code의 복호에는 사용할 수 없다. Viterbi 알고리즘에서 각 state에 입력되는 누적된 path metric의 차이인 difference metric을 사용하여 각 비트에 대한 reliability information을 계산하여 soft decision value를 출력하는 알고리즘인 SOVA가 제안되었다[4]. SOVA는 surviving path와 이와 합쳐지는 competing path만을 이용하여 soft decision value를 출력하기 때문에 모든 path를 고려하여 reliability information을 계산하는 MAP을 기반으로 하는 알고리즘에 비해서 성능이 다소 떨어지지만 계산량은 줄어든다. MAP을 기반으로 하는 알고리즘 중에서 계산량이 가장 적은 Max-Log-MAP 알고리즘에 비해서 SOVA의 계산량은 절반 정도이다[3]. 따라서 SOVA는 멀티미디어 정보와 같이 대용량의 정보를 실시간으로 처리해야 하는 응용에 적합한 알고리즘으로 볼 수 있다. SOVA를 변형하여 surviving path와 합쳐지는 competing path 뿐 만 아니라 competing path와 합쳐지는 모든 path도 고려하여 출력을 계산하는 modified SOVA도 제안되었다. Modified SOVA는 결과적으로 모든 path를 고려하여 출력을 계산하는 것이 되고 Max-Log-MAP 알고리즘과 동일한 성능을 보인다[5]. Modified SOVA를 구현하기 위해서는 competing path와 합쳐지

는 모든 path도 고려해야 하므로 SOVA에 비해서 log-likelihood ratio 갱신 횟수가 늘어나게 되고 trace-back 연산이 크게 늘어나게 된다.

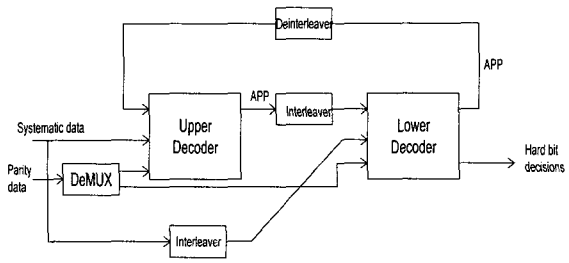

그림 1. Turbo code의 복호기 구조

SOVA는 순방향으로 state metric과 difference metric을 계산하는 부분과 최적 sequence인 surviving path를 찾는 부분 및 difference metric을 이용하여 reliability 값을 계산하는 log-likelihood ratio 갱신 부분으로 구성된다. Berrou 등은 SOVA를 효율적으로 구현하는 구조를 제시하였는데[8], 이는 다음과 같은 과정으로 구현된다.

1. 매 stage k마다 각 state에 입력되는 두 개의 path에 대해서 최대의 path metric을 그 state의 state metric으로 하고 각 state에 대해서 surviving path에 대한 정보와 두 path metric의 차이인 difference metric을 저장한다.
2. Stage k에서 k가 $q(\geq 5\times v$, v는 code memory)보다 큰 경우 최대의 state metric을 갖는 state S_k^{MaxMet}을 선택한다. S_k^{MaxMet}로부터 출발하여 l 시간 전까지 trace-back을 수행하여 k-l stage에서 최적인 state S_{k-l}^{MaxLP}를 추정한다.
3. 2번의 과정에서 선택된 state S_{k-l}^{MaxLP}에 입력되는 surviving path의 accumulated metric M_{k-l}^{s}과 competing path의 accumulated metric M_{k-l}^{c}의 차이인 difference metric Δ_{k-l}가 reliability value가 된다. Surviving path와 competing path 상으로 두 path가 합쳐질 때까지 trace-back을 수행하면서 surviving path 상의 u_j^s와 competing path 상의 u_j^c가 같지 않다면 다음 식과 같이 각 bit에 대한 log likelihood ratio L_j를 갱신한다.

$$L_j = sign(u_j^s) \cdot \min(\Delta_{k-l}, L_j) \qquad (1)$$

1~3의 과정을 매 bit가 입력될 때마다 반복한다.

Ⅲ. 제안하는 SOVA

앞에서 설명한 SOVA를 그림 2의 state 수가 4인 trellis 도에서 설명하면 다음과 같다. 편의상 모든 bit가 0인 경우인 state *S0*를 따르는 sequence를 maximum likelihood sequence로 가정하였고 이를 가장 굵은 선으로 표시하였다. 또한 maximum likelihood path 상의 각 stage의 각 state에 입력되는 competing path를 중간 굵기의 선으로 표시하였다. 그림에서 surviving path가 되는 maximum likelihood sequence 상의 k번째 bit인 u_k^s (S는 surviving path 상의 bit를 의미하고 k는 k번째 stage를 의미함) 에 대한 log-likelihood ratio를 결정하기 위해서는 k~$k+3$번째 시간에서 surviving path와 합쳐지는 competing path 상의 bit인 $u_{k(k)}^c \sim u_{k(k+3)}^c$ (C는 competing path, k는 k번째 stage 상의 bit를 의미하고 괄호안의 k~$k+3$은 competing path가 k~$k+3$번째 stage에서 surviving path와 합쳐짐을 의미함) 이 모두 u_k^s와 다르므로 $k \sim k+3$번째 difference metric인 $\Delta_k, \Delta_{k+1}, \Delta_{k+2}, \Delta_{k+3}$ 중에서 최소 값이 log likelihood ratio L_k가 된다. 따라서 기존의 SOVA로 L_k 값을 결정하기 위해서는 $\Delta_k, \Delta_{k+1}, \Delta_{k+2}, \Delta_{k+3}$의 각 값과 log-likelihood ratio 값인 L_k을 비교해서 log-likelihood 값을 update하는 식 (1)의 연산을 4번 수행해야 한다. 그런데 trace-back 연산 중에서 $k+2$, $k+3$에 대응하는 competing path는 stage k의 state *S2*(검은 원으로 표시)에서 합쳐진 이후에 stage k-*1*에서 surviving path와 합쳐진다. 이러한 경우에 L_k의 update에는 Δ_{k+2}, Δ_{k+3} 중에서 작은 값만 유효하다. 만일 competing path가 합쳐질 때 각 competing path에 대응하는 difference metric 중에서 작은 값 만을 고려한다면 log-likelihood ratio 갱신의 횟수를 줄일 수 있게 된다.

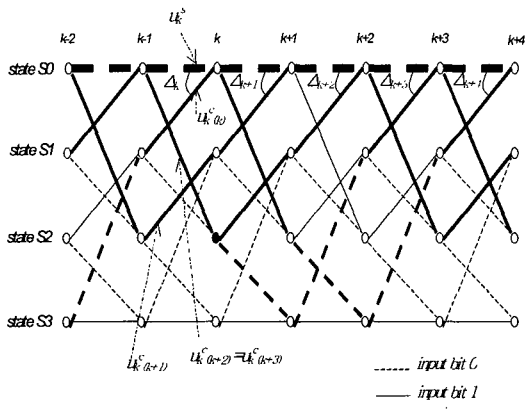

그림 2. State 수가 4인 trellis 도의 보기

이러한 점을 고려하여 log-likelihood ratio의 갱신을 병렬적으로 구현하는 다음과 같은 방법을 제안한다. 먼저 각 stage k의 Δ_k 값을 competing path 상에서 시간의 역순으로 trace-back하는 과정에서 거치게 되는 state에 병렬적으로 할당하고 competing path가 동일한 state에서 합쳐지게 되는 경우에는 작은 값만을 선택하여 state에 할당한다. Log-likelihood ratio의 갱신에는 state에 할당된 Δ_k를 사용하기 때문에 각 state에서 선택되지 않은 값에 대해서는 식 (1)의 update하는 연산을 수행하지 않게 되어 전체 log-likelihood ratio 갱신의 수를 줄일 수 있게 된다. 예를 들면 그림 2의 k번째 시간에서 state $S0$는 surviving path 상의 state가 되고 state $S1$에는 Δ_{k+1} 값이 할당되고 state $S2$에는 Δ_{k+2}, Δ_{k+3} 중에서 작은 값이 할당되게 된다. 또한 state $S3$에는 아무 값도 할당되지 않는다. 이러한 방법을 사용하면 L_k의 계산을 위해서 Δ_{k+2}, Δ_{k+3} 중에서 큰 값과의 비교는 제외된다.

그런데 제안하는 알고리즘을 전체 블록에 대해서 적용한다면 difference metric 값의 저장과 trace-back을 위한 정보의 저장에 매우 많은 기억장소를 요구한다. 이러한 문제를 해결하기 위해서 Log-MAP 알고리즘에 적용하는 sliding window 알고리즘[6]과 같이 전체 블록을 일정한 크기의 부블럭으로 나누어서 log-likelihood ratio를 구하는 방법을 제안하는 알고리즘에 적용하였다. 이러한 방법을 적용하면 trace-back도 부블럭 단위로 수행되므로 trace-back을 위한 연산도 줄일 수 있게 된다. 위에서 설명한 원리를 바탕으로 본 논문에서 제안하는 효율적인 SOVA는 다음과 같은 과정으로 구현할 수 있다.

1. 전체 블록($0 \le k \le L-1$)을 일정한 크기 N인 P $(0 \sim P-1)$ 개의 부블럭으로 나눈다.
2. 각 부블럭에 대해서 forward recursion을 수행하여 각 state에 대한 difference metric과 surviving path에 대한 정보를 저장한다.
3. $l(\ge 1)$번째 부블럭에 대해서 2번의 과정을 종료한 후에 부블럭의 마지막 시간 $(l+1)N-1$에서 최대의 metric 값을 갖는 state로부터 trace back을 수행하여 이전 부블럭의 마지막 시간 $lN-1$의 maximum -likelihood state S_{lN-1}^{MaxLP}를 추정한다.
4. S_{lN-1}^{MaxLP} 부터 출발하여 surviving path의 trace-back을 수행하는 것과 동시에 각 시간에서 모든 competing path 상의 이전 state에 difference metric 값을 할당한다. 만일 state에 difference metric 값이 이미 할당되어 있는 경우라면 두 값을 비교하여 작은 값을 state의 difference metric으로 한다.
5. 3번의 각 과정에서 difference metric이 할당되고 survivor path의 이전 bit u_j^s와 competing path 상의 state의 이전 bit u_j^c를 비교하여 두 bit이 같지 않은 경우에는 state에 할당된 difference metric을 식 (1)의 Δ_{k-l}로 하여 log-likelihood ratio를 갱신한다.
6. 3~5의 과정을 $l-1$번째 부블럭에 대해서 $lN-1$부터 $(l-1)N$ 까지 역 방향으로 수행한다. 부블럭의 경계 부분인 $(l-1)N-1$부터 $(l-1)N-b$ 구간에서는 $\Delta_{(l-1)M-1} \sim \Delta_{(l-1)M-b}$를 사용한 log-likelihood ratio의 갱신은 이전 부블럭의 계산에서 이미 수행되었으므로 $(l-1)N-1$부터 $(l-1)N-b$ 구간에 대해서는 $l-1$번째 부블럭에서 전파되어온 state difference metric 갱신과 이를 이용한 log-likelihood ratio의 갱신만을 행한다.

2~6의 과정을 전체 부블럭에 대해서 반복한다.

제안하는 알고리즘의 4번 과정에서 알 수 있는 것과 같이 각 부블럭 별로 trace-back 연산이 동시에 진행되기 때문에 trace-back 연산 횟수가 기존의 SOVA에 비해서 줄어든다. 또한 6번의 과정은 부블럭 간의 연속성을 보장하기 위한 것이다. 이것은 Log-MAP을 위한 sliding window 알고리즘에서 각 부블럭 끝부분의 경계값에서 초기 β 값을 구하기 위한 과정에 대응한다고 생각할 수 있다.

Ⅳ. 모의 실험

3장에서 제안한 방법을 이용하여 SOVA를 구현하였을 때의 성능과 계산량을 기존의 SOVA와 비교하기 위해서 3GPP 표준에서 사용되는 state 수가 8인 turbo code[7]에 대해서 모의 실험을 수행하였다. 3GPP 표준 turbo code의 부호화기의 블록도를 그림 3에 도시하였다. 모의 실험에서 turbo code의 iteration 수를 계산 시간을 고려하여 8로 제한하였고 제안하는 알고리즘에서 부블럭의 크기는 128로 하였다. 또한 turbo code의 부호화율을 조절하기 위해서는 parity 비트를 펑처링할 수 있는데 RC(rate compatible) 특성을 만족하도록 parity 비트를 펑처링하여 부호화율을 1/2과 2/3로 한 경우에 AWGN 채널 환경에서 모의실험을 수행하였다. 성능 평가를 위하여 BER(bit error rate) 결과를 제시하였고 계산량의 비교를 위하여 SOVA에서 가장 많은 계산량을 차지하는 log-likelihood ratio 갱신의 수를 구하여 이를 비교하였다.

그림 4에는 E_b/N_0가 변할 때의 BER 성능을 제시하였고 그림 5에는 block size가 변할 때의 BER 성능을 제시하였다. Sliding window 방법을 사용하지 않고 전체 블록에 대해서 제안하는 방법을 적용하였을 때는 기존의 방법과 동일한 성능을 보인다. 그림 4와 그림 5는 sliding window 방법을 적용하였을 때의 결과를 제시한 것인데 거의 구분할 수 없을 정도의 성능 차이를 보이는 것은 제안하는 방법에서 sliding window 알고리즘을 적용할 때 각 부블럭에서 겹쳐지는 정도를 나타내는 b (3장에서 제시하는 알고리즘 step 6의 b) 값에 따라서 생기는 성능의 차이인데 그림 4와 그림 5의 결과는 b 값을 40으로 하였을 때이다. 제안하는 방법과 기존의 방법 모두 블록 크기가 증가함에 따라서 BER이 크게 감소하는 turbo code의 특성을 보인다. 그림 6과 그림 7에는 계산량의 비교를 위하여 log-likelihood ratio 갱신의 수를 제시하였다. 식 (1) 계산의 평균 횟수를 비교한 것인데 제안하는 방법이 기존의 방법에 비해서 65~75% 정도의 계산 만을 요구하는 것을 알 수 있다. 기존의 알고리즘은 surviving path와 competing path가 합쳐지는 정도가 달라질 수 있고 이는 log-likelihood ratio의 갱신에 직접 영향을 미치므로 평균 갱신 횟수가 다소 변한다. 이에 비해서 제안하는 알고리즘은 병렬적인 구현 방법을 사용하기 때문에 log-likelihood ratio 갱신의 횟수가 거의 변하지 않는다. 또한 제안하는 방법은 부블럭 내에서 trace-back을 동시에 수행하므로 각 시간에서 trace-back을 수행하는 기존의 알고리즘에 비해서 trace-back을 위한 시간을 줄일 수 있고 surviving path와 competing path에 대한 비교도 모든 competing path에 대해서 동시에 진행하므로 효율을 높일 수 있다. 제안하는 알고리즘은 기존의 SOVA에 비해서 surviving path와 competing path에 대한 trace-back 연산이 1/10 정도로 줄어든다. 다만 기존의 SOVA에 비해서 각 state에 metric을 할당하기 위한 별도의 계산이 필요하다.

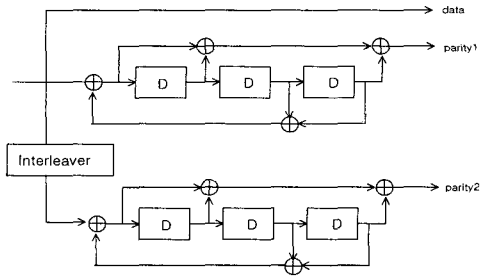

그림 3. 3GPP 표준의 turbo encoder 블록도

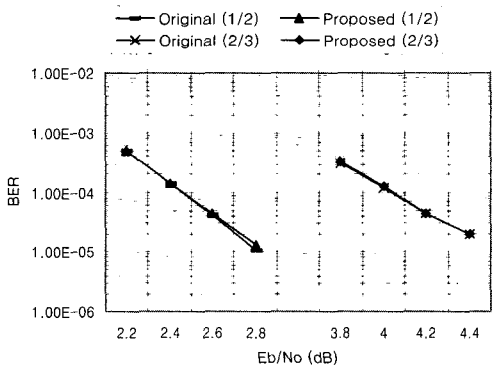

그림 4. AWGN 채널에서 E_b/N_0가 변할 때 기존의 SOVA와 제안하는 SOVA의 BER 비교 (block size: 512, channel code rate:1/2 혹은 2/3, turbo code 복호를 위한 iteration 수: 8)

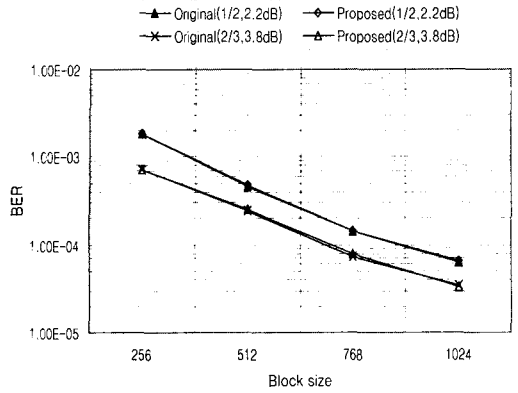

그림 5. AWGN 채널에서 block size가 변할 때 기존의 SOVA와 제안하는 SOVA의 BER 비교 ((E_b/N_0:2.2dB, channel code rate:1/2) 혹은 (E_b/N_0:3.8dB, channel code rate:2/3), turbo code 복호를 위한 iteration 수: 8)

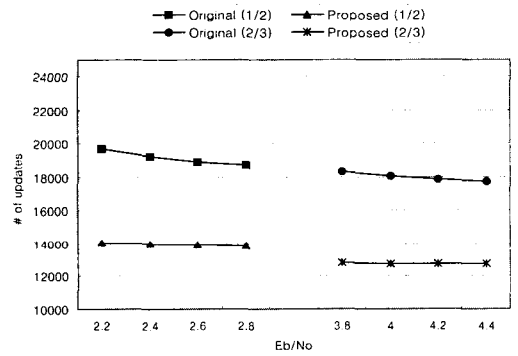

그림 6. 기존의 SOVA와 제안하는 SOVA의 계산량 비교 (그림 4과 같은 조건, log-likelihood ratio 갱신의 평균 횟수 비교)

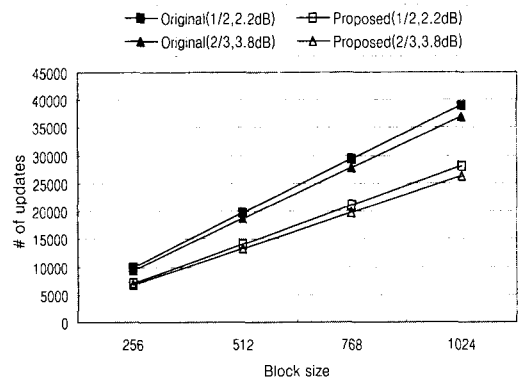

그림 7. 기존의 SOVA와 제안하는 SOVA의 계산량 비교 (그림 5와 같은 조건, log-likelihood ratio 갱신의 평균 횟수 비교)

V. 결 론

본 논문에서는 turbo code를 포함한 concatenated code의 복호에 많이 사용되는 SOVA를 효율적으로 구현하는 알고리즘을 제안하였다. 이를 위하여 기존의 알고리즘에서는 각 stage 별로 독립적으로 진행되는 log-likelihood ratio의 갱신 과정을 병렬적으로 구현하여 전체의 계산량을 감소시키는 방법을 제안하였다. 또한 블록의 크기가 커질 때 메모리의 사용이 크게 증가한다는 점을 고려하여 memory를 효율적으로 사용할 수 있는 sliding window 형태를 사용하여 전체 알고리즘을 구현하였다.

성능 평가를 위해서 제안하는 SOVA를 3GPP 표준에서 사용하는 turbo code에 적용한 결과 기존의 SOVA와 동일한 성능을 보이면서 SOVA의 핵심적인 부분인 log-likelihood ratio 갱신의 횟수는 기존의 알고리즘의 65%~75% 만을 필요로 한다는 것을 알 수 있었다. 또한 부블럭 내의 trace-back이 동시에 진행되고 각 competing path와 surviving path에 대한 비교도 모든 competing path에 대해서 동시에 수행되므로 계산량은 더욱 줄어든다. Turbo code의 많은 계산량을 고려해 볼 때 제안하는 알고리즘은 기존의 알고리즘 중에서 가장 계산량이 적은 SOVA를 더욱 효율적으로 구현한 알고리즘이므로 멀티미디어 정보와 같은 대용량의 데이터를 고속으로 처리하기에 적합한 알고리즘으로 판단된다.

참 고 문 헌

[1] C. Berrou, A. Glavieux and P. Thitimajshima, "Near Shannon limit error correcting coding and decoding: turbo-codes," *IEEE Proc. ICC '93*, Geneva, Switzerland, May 1993, pp. 1064-1070.

[2] L. Hanzo, T. H. Liew and B. L. Yeap, *Turbo coding, turbo equalization and space-time coding*, John Wiley & Sons, 2002.

[3] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," in *Proc. Int. Conf. Communications*, June 1995, pp. 1009-1013.

[4] J. Hagenauer and P. Hoeher, "A Viterbi

algorithm with soft-decision outputs and its applications," *IEEE Globecom*, 1989, pp. 1680-1686.

[5] M. P. C. Fossorier, F. Burkert, S.Lin and J. Hagenauer, "On the equivalencebetween SOVA and Max-Log-MAP decodings," *IEEE Commun. Letters*, vol. 2, pp.137-139, 1998.

[6] A. J. Viterbi, "An intuitive justification and simplified implementation of the MAP decoder for convolutional codes," *IEEE J. Sel. Areas in Comm.* vol. 16, pp.260-264, Feb. 1998.

[7] 3rd Generation Partnership Project, "Multiplexing and channel coding(FDD)," *3GPP Technical Specification*, TS 25.212 v4.2.0, Sep. 2001.

[8] C. Berrou, P. Adde, E. Angui, and S. Faudeil, "A low complexity soft-output Viterbi decoder architecture," in *Proc. Int. Conf. Communications*, May 1993, pp.737-740.

이 창 우 (Chang-Woo Lee) 정회원
한국통신학회 논문지 제25권 제7호 참조
현 재 : 가톨릭대학교 정보통신전자공학부 부교수

<주관심분야> 영상 압축, 영상 전송, turbo 부호