블록 통합을 사용한 효율적 터보 디코더 설계

서종현, 윤상훈, 정정화
한양대학교 전자통신전파공학과 CAD 및 통신회로 연구실
전화 : 02-2290-0558 / 핸드폰 : 019-431-3719

An efficient method for Turbo Decoder design using Block Combining

Jong-Hyun Seo, Sang-Hun Yoon, Jong-Wha Chong
Dept. of CAD & communication circuit, Hanyang University
E-mail : firsthero@hanmail.net

Abstract

본 논문에서는 터보 디코더에 사용되는 MAP 알고리즘의 저전력 구조를 제안한다. 터보 디코더 알고리즘 중 하나인 MAP 알고리즘은 많은 메모리 사이즈와 복잡한 연산량을 가진다. 본 논문에서는 메모리 사이즈를 줄이기 위하여 두 번의 상태 천이(branch metric) 과정을 하나로 통합 계산하는 방식을 제안하였다. 제안된 방식으로 구한 상태 천이 값을 이용해서 FSM(Forward State Metric)값을 구하면 BM(branch metric)값이 다음 상태의 FSM에 포함되어지므로 APP(A Posteriori Probability)를 계산할 때 BM부분이 빠져 LLR(Log Likelihood Ratio)의 연산량을 줄일 수 있다. 실험결과 기존의 MAP 알고리즘과 동일 성능을 가지면서 MAP 알고리즘을 개선한 Pietrobon 알고리즘을 log-MAP 알고리즘에 적용하여 LLR 연산량을 비교했을 때 덧셈 연산을 반으로 줄일 수 있음을 확인하였다.

I. 서론

통신 시스템이 발전함에 따라 정보 전송이 고속의 신뢰성 있는 채널 코딩 방식이 요구된다. 그러나 통신 채널에서 잡음이나, 페이딩, 간섭 등에 의한 정보 손실이 발생하게 되고, 이로 인해 발생하는 오류를 정정해 주기 위해 오류정정부호의 사용이 필요하다.

오류정정부호의 연구는 1948년 Shannon 연구결과 발표 이후 많은 곳에서 연구되어지고 있다. 그중 터보코드는 1993년 Berrou, Glavieux, Thitimajshima 에 의해 제안되어졌고[1], Shannon의 한계에 근접하는 매우 우수한 오류정정 능력을 제공함으로써 이에 대한 연구가 활발하게 이루어지고 있다.

터보 코드의 복호화는 MAP(Maximum a Posteriori) 복호기 혹은 SOVA(Soft-Output Viterbi Algorithm) 복호기를 이용한 반복 복호를 통하여 원래의 정보를 복원하게 된다. MAP 알고리즘은 SOVA 알고리즘에 비해 복잡성은 크나, 성능면에서 우수성을 가진다[3]. 이러한 이유로 복잡성은 크나, BER 성능이 우수한 MAP 알고리즘이 사용되고 있다[4]. MAP 알고리즘은 Bahl 등에 의해서 1974년 처음으로 제안되어 졌으며, 잡음이 섞인 신호로부터 APP를 계산하는 알고리즘이다. MAP 알고리즘은 모든 경로에 대한 확률 값들을 계산하여 정보비트에 대한 연판정 데이터 값을 출력하게 된다.

본 논문의 구성은 다음과 같다. 2장에서 MAP 알고리즘을 설명하고 3장에서 블록 프로세싱을 이용한 알고리즘에 대해서 기술한다. 4장에서 블록 통합을 사용한 효율적인 MAP 알고리즘을 설명하고, 5장에서 실험 결과 및 고찰을 기술한다. 마지막으로 6장에서 결론을 맺도록 한다.

II. MAP 알고리즘

MAP 알고리즘은 Bahl 등에 의해 1974년 처음 제안되었으며 잡음이 섞인 신호로부터 APP를 계산하는 알고리즘이다[4][5]. MAP 디코딩 알고리즘의 목적은 데

이터가 수신된 후 수신된 심볼에 대해서 가장 확률이 큰 정보 비트를 결정하게 된다. 이는 LLR(Log Likelihood Ratio)라고 한다. 그림1은 4-state의 시간 k-1에서 k+1까지의 trellis diagram을 보여주고 있다. 여기서 alpha는 순방향 상태 메트릭(forward state metric)이라 하며, 이것은 정보비트를 가지고 시간 k-1의 이전 상태 (s')부터 시간 k에 대한 다음 상태(s)로의 천이를 위한 상태 메트릭을 나타낸다. beta은 역방향 상태 메트릭(Backward state metric)이라고 하며, alpha과 마찬가지로 모든 정보를 수신한 후, 이전상태의 beta값에서 현재의 beta값을 반복적으로 구할 수 있다. gamma는 가지 메트릭(Branch metric)으로 정의되어진다.

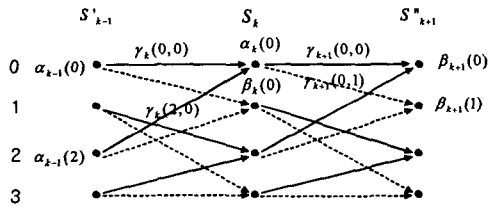

그림 1. 4-state의 시간 k-1에서 k+1까지의 trellis diagram

LLR을 계산하기 위해서는 먼저 받은 데이터를 가지고 gamma 값을 구하고, gamma값을 이용해서 alpha값과 beta값을 구하게 된다. 계산식은 아래와 같다.

$$\alpha_k(s) = \sum_{all\_s'} \gamma_k(s',s) \cdot \alpha_{k-1}(s') \quad (1)$$

$$\beta_{k-1}(s') = \sum_{all\_s'} \beta_k(s) \cdot \gamma_k(s',s) \quad (2)$$

$$\gamma_k^{s',s} = \exp\{\frac{2}{\sigma^2}(x_k u_k + y_k v_k)\} \quad (3)$$

u_k 는 데이터 비트이고, v_k 는 패리티 비트이다. x_k 는 채널을 통과한 노이즈가 섞인 데이터 비트이고, y_k 는 채널을 통과한 노이즈가 섞인 패리티 비트이다.

alpha, beta, gamma 값을 구했으면 이 값들을 가지고 LLR을 다음과 같이 계산하게 된다.

$$L(u_k) = \ln\left(\frac{\sum_{\substack{(s',s)=>\\ u_k=+1}} \alpha_{k-1}(s')\gamma_k(s',s)\beta_k(s)}{\sum_{\substack{(s',s)=>\\ u_k=-1}} \alpha_{k-1}(s')\gamma_k(s',s)\beta_k(s)}\right) \quad (4)$$

$$L(u_{k+1}) = \ln\left(\frac{\sum_{\substack{(s,s'')=>\\ u_{k+1}=+1}} \alpha_{k-1}(s)\gamma_k(s,s'')\beta_k(s'')}{\sum_{\substack{(s,s'')=>\\ u_{k+1}=-1}} \alpha_{k-1}(s)\gamma_k(s,s'')\beta_k(s'')}\right) \quad (5)$$

식(4),(5)는 시간 k, k+1에서 각각 LLR을 계산한 식이다. 식(4),(5)의 의미는 이전상태에서 현재 상태로 넘어올 때 확률이 가장 큰 정보 비트를 결정하는 것이다.

III. 블록 프로세싱을 이용한 알고리즘

본 장에서는 메모리를 효율적으로 사용하기 위해 제안된 블록 프로세싱 기법을 이용한 알고리즘에 대해서 설명한다[2].

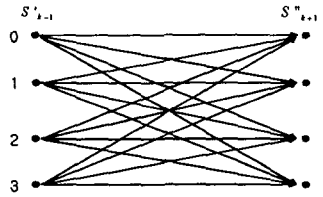

그림 2. 4-state의 시간 k-1에서 k+1까지의 블록 프로세싱 기법을 적용한 trellis diagram

블록 프로세싱 기법을 이용한 알고리즘은 시간 k-1에서 시간 k+1까지의 alpha값과 beta값을 시간 k에 대해서 계산하지 않고 k+1에서 계산함으로 중간 과정인 시간 k에서의 alpha값과 beta값을 저장할 메모리를 줄였다. alpha값과 beta값을 구하는 수식은 아래와 같다.

$$\alpha_{k+1}(s) = \sum_{all\_s'} \gamma(s',s,s'') \cdot \alpha_{k-1}(s')$$

$$\beta_{k-1}(s') = \sum_{all\_s''} \gamma(s'',s,s') \cdot \beta_{k+1}(s'')$$

$$\gamma(s'',s,s') = \gamma(s'',s) \cdot \gamma(s,s')$$

블록 프로세싱 기법을 이용한 알고리즘에서는 기존의 MAP 알고리즘에서 효율적인 메모리 사용으로 데이터 억세스의 횟수를 줄임으로 전력을 감소시켰으나, LLR을 계산할 때 기존 MAP 알고리즘에 비해 곱셈연산이 많아지는 단점이 있다.

IV. 블록 통합을 사용한 효율적인 MAP 알고리즘

본 장에서는 블록 통합을 사용한 효율적 MAP 알고리즘을 설명한다.

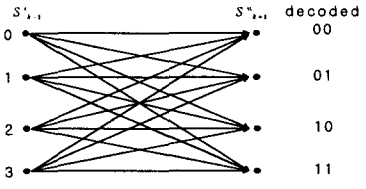

그림 3. 4-state의 시간 k-1에서 k+1까지의 블록 통합 trellis diagram

그림3에서는 두 번의 디코딩 과정을 한번으로 통합하여 디코딩 하는 과정을 보여주고 있다. 각 상태별로 통합 계산된 BM값을 가지고 FSM, BSM을 구한다. 각 상태별로 통합된 BM 값의 의미는 시간 k-1에서 시간 k까지의 BM 값과 시간 k에서 k+1까지의 BM값을 통합 계산된 것이다. 수식은 아래와 같다.

$$\gamma_k^{s',s,s''} = \gamma_k^{s',s} \times \gamma_k^{s,s''}$$

$$= \exp\{\frac{2}{\sigma^2}(x_k u_k + y_k v_k + x_{k+1}u_{k+1} + y_{k+1}v_{k+1})\} \quad (6)$$

$$\beta_{k-1}(s') = \sum_{all\_s''} \beta_k(s'') \cdot \gamma_k(s', s'') \quad (7)$$

$$\alpha_{k+1}(s) = \sum_{all\_s'} \gamma_k(s', s'') \cdot \alpha_{k-1}(s') \quad (8)$$

$\gamma_k^{s',s''}$ 값은 시간 k-1의 상태(S')에서 시간 k의 상태(S)로 천이한 다음, 시간 k의 상태(S)에서 다음 상태(S'')로 천이된 것을 통합 계산한 값이다. 통합된 BM값을 가지고 시간 k+1의 alpha값과 beta값을 구하게 된다. 여기서 alpha값과 beta값을 구하는 것은 기존 MAP 알고리즘의 계산방식과 동일하다. 즉, alpha은 시간 k-1의 모든 state에서 시간 k+1의 어느 한 state로 천이될 상태 메트릭을 나타내고 있다. beta도 마찬가지로 이전 상태의 beta값에서 반복적으로 구한다.

시간 k-1의 모든 state에서 시간 k+1의 '0' state 로 천이될 경우는 데이터를 (0,0)을 수신했을 경우이고, 시간 k+1의 '1' state로 천이될 경우는 데이터를 (0,1)을 수신한 경우이다. 그리고 시간 k+1의 '2' state로 천이될 경우는 데이터를 (1,0)을 수신했을 경우가 되고, 시간 k+1의 '3' state로 천이될 경우는 데이터를 (1,1)을 수신했을 경우가 된다. 각각의 state로 천이될 경우에 대한 확률값을 계산하면 아래식과 같다.

$$p(u_k = 00 : y_k) = \sum_{0,0} \alpha_{k-1}\gamma_{s',s''}\beta_{k+1} \quad (9)$$

$$p(u_k = 01 : y_k) = \sum_{0,1} \alpha_{k-1}\gamma_{s',s''}\beta_{k+1} \quad (10)$$

$$p(u_k = 10 : y_k) = \sum_{1,0} \alpha_{k-1}\gamma_{s',s''}\beta_{k+1} \quad (11)$$

$$p(u_k = 11 : y_k) = \sum_{1,1} \alpha_{k-1}\gamma_{s',s''}\beta_{k+1} \quad (12)$$

$p(u_k = 00 : y_k)$는 데이터 y_k을 받았을 때 받은 데이터 값이 (0,0)일 확률을 나타내고, $p(u_k = 01 : y_k)$는 데이터 y_k을 받았을 때 받은 데이터가 (0,1)일 확률을 나타낸다. 그리고 $p(u_k = 10 : y_k)$는 (1,0)일 확률, $p(u_k = 11 : y_k)$는 (1,1)일 확률을 나타낸다.

각각에 대한 확률값을 구하고 나면 이들 중 최대값을 결정하게 된다. 최대값을 결정하는 이유는 시간 k-1에서 k+1로 천이될 때의 경로를 결정해서 LLR을 계산할 때 최대값에 따라 달라지는 계산과정 때문이다. 최대값에 따른 LLR의 계산 과정은 아래와 같다.

case1($p(u_k = 00 : y_k)$)

$$L(u_k) = \log(\frac{p(u_k = 10 : y_k)}{p(u_k = 00 : y_k)}), \quad L(u_{k+1}) = \log(\frac{p(u_k = 01 : y_k)}{p(u_k = 00 : y_k)}) \quad (13)$$

case2($p(u_k = 01 : y_k)$)

$$L(u_k) = \log(\frac{p(u_k = 11 : y_k)}{p(u_k = 01 : y_k)}), \quad L(u_{k+1}) = \log(\frac{p(u_k = 01 : y_k)}{p(u_k = 00 : y_k)}) \quad (14)$$

case3($p(u_k = 10 : y_k)$)

$$L(u_k) = \log(\frac{p(u_k = 10 : y_k)}{p(u_k = 00 : y_k)}), \quad L(u_{k+1}) = \log(\frac{p(u_k = 11 : y_k)}{p(u_k = 10 : y_k)}) \quad (15)$$

case4($p(u_k = 11 : y_k)$)

$$L(u_k) = \log(\frac{p(u_k = 11 : y_k)}{p(u_k = 01 : y_k)}), \quad L(u_{k+1}) = \log(\frac{p(u_k = 11 : y_k)}{p(u_k = 10 : y_k)}) \quad (16)$$

$L(u_k)$는 시간 k에서의 디코딩된 값이 되고, $L(u_{k+1})$는 시간 k+1에 대한 디코딩된 값이 된다. case1의 $L(u_k)$에서 $p(u_k = 10 : y_k)$을 $p(u_k = 00 : y_k)$로 나누어 주는 것은 $p(u_k = 00 : y_k)$이 시간 k-1에서 k로 가는데 0을 받아서 천이되는 확률값과 시간 k에서 k+1로 가는데 0을 수신해서 천이되는 확률값이 곱해 진 것이기 때문에 시간 k+1에서 디코딩한 값을 상쇄시켜 주어야 한다. 그러기 위해서 식(13)과 같이 $p(u_k = 10 : y_k)$을 $p(u_k = 00 : y_k)$로 나누어 주었다. 이렇게 함으로써 $L(u_k)$에 대한 디코딩된 값을 얻어 낼 수 있다. $L(u_{k+1})$도 마찬가지로 시간 k-1에서 k로 가는 확률값을 상쇄시켜 줌으로 해서 $L(u_{k+1})$을 얻어 낼 수 있다. case2일 때, case3일 때 그리고

case4일 때도 마찬가지로 $L(u_k)$, $L(u_{k+1})$을 계산하게 된다. 식(9)은 아래와 같이 좀 더 간략화 시킬 수 있다.

$$p(u_k = 00 : y_k) = \sum_{0,0} \alpha_{k-1} \gamma_{s',s} \beta_{k+1} \quad (17)$$

$$= \alpha_{k-1}(0) \cdot \gamma_{0,0} \cdot \beta_{k+1}(0) + \alpha_{k-1}(1) \cdot \gamma_{1,0} \cdot \beta_{k+1}(0) + \alpha_{k-1}(2) \cdot \gamma_{2,0} \cdot \beta_{k+1}(0) + \alpha_{k-1}(3) \cdot \gamma_{3,0} \cdot \beta_{k+1}(0)$$
$$= (\alpha_{k-1}(0) \cdot \gamma_{0,0} + \alpha_{k-1}(1) \cdot \gamma_{1,0} + \alpha_{k-1}(2) \cdot \gamma_{2,0} + \alpha_{k-1}(3) \cdot \gamma_{3,0}) \times \beta_{k+1}(0)$$
$$= \alpha_{k+1}(0) \times \beta_{k+1}(0)$$

식(16)에서 수신한 데이터가 (0,0)일 확률값은 시간 k+1의 alpha값이 이전상태의 모든 state에 대한 alpha 값을 포함하고 있으므로 현재 상태의 alpha와 beta만으로 수식이 간략화 될 수 있다.

식(10),(11)(12)도 동일하게 간략화 시킬 수 있다.

V. 실험 및 고찰

log-MAP 알고리즘에 의해 곱셈 연산을 덧셈 연산으로 바뀔 수 있다[7]. 하드웨어 구현을 용이하게 하기 위해 log-MAP 알고리즘을 적용하고, 기존의 알고리즘과 제안하는 알고리즘에 대해 LLR 계산에서 사용되는 연산량과 메모리수를 비교한다.

표 1. 제안된 알고리즘과 기존 알고리즘들을 log-MAP 알고리즘을 적용 하여 LLR 계산에서 연산량과 메모리(BM, FSM, BSM) 비교(n:상태수, m:데이터크기)

	log-MAP 알고리즘	Pietrobon 알고리즘	블록프로세싱 알고리즘	제안된 알고리즘
덧셈 연산	4*n(m-1)	n(m-1)	12*n((m-1)/2)	1/2*n(m-1)
max 연산	3/2*n(m-1)	1/2*n(m-1)	3*n(m-1)	(n-1)((m-1)/2)
FSM(or BSM) 메모리	m*n	m*n	1/2*n(m+1)	1/2*n(m+1)
전체 메모리수	2*n(2*m-1)	2*n(2*m-1)	n(3*m-1)	n(3*m-1)

제안된 알고리즘은 기존 알고리즘들에 비해 적은 덧셈 연산이 수행되고, 메모리 수에서 블록 프로세싱과 같은 수를 가지나 연산량이 훨씬 적은 것을 알 수 있었다. 따라서 제안된 방법에 의하여 디코더를 구성할 경우 훨씬 적은 덧셈기가 필요 되어 전체 회로 면적을 줄일 수 있음을 알 수 있었다.

VI. 결론

터보 디코딩 알고리즘중 하나인 MAP 알고리즘은 우수한 오류정정 능력을 가지고 있으나 복잡성이 크다. 이러한 복잡성을 줄이기 위해 본 논문에서는 블록통합을 사용한 효율적 MAP 알고리즘을 제안하고 있다. 블록 통합에서는 두 번의 디코딩 과정을 하나로 통합해서 디코딩 하게 되고 그 결과 제안된 알고리즘에서는 메모리수과 연산량을 줄이고 있다.

그러므로 제안된 블록 통합에 의한 MAP 알고리즘은 효율적인 메모리 사용과 저전력 구현이 가능하므로, 고속 통신 및 저전력 특성을 요구하는 통신 시스템에 적합하다고 할 수 있다.

참고문헌

[1] C. Berrou, A. Glavieux, P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-Codes" in *Proceeding of IEEE International Conferences on Communications '93,* May 1993, pp. 1064-1070.

[2] Lee, I. Vallejo, M.L. Mujtaba, S.A. "Block Processing Technique for Low Power Turbo Decoder Design" *VTC Spring 2002. IEEE 55th,* pp.1025-1029.

[3] Papke, L., Robertson, P. and Villebrun, E., "Improved decoding with the SOVA in a parallel concatenated (trubo-code) scheme" *Proceeding of ICC '96, Dallas, TX, USA,* June, 1996, pp. 102-106.

[4] Bahl. L. Cocke. J. Jelinek, F. and Raviv. J,. "Optimal decoding of linear codes for minimizing symbol error rate." *IEEE Trans. Inform. Theory,* vol. IT-20, Mar. 1974. pp. 284-287.

[5] Pietrobon, S. and Barbulescu, A.S, "A simplification of the modified Bahl decoding algorithm for systematic convolutional codes," *Proceeding of ISITA'94, Sydney, Australia,* Nov. 1994, pp. 875-880.

[6] Pietrobon, S., Barbulescu, A.S, "A simplification of the modified Bahl decoding algorithm for systematic convolutional codes," *Proceeding of ISITA'94, Sydney, Australia,* Nov. 1994, pp.875-880.

[7] P.Robertson,E. Villebrun, and P.Hoher, "A Comparison of Optimal and Sub-Optimal MAP Decoding Algorithms Operating in the Log Domain," in *Proceedings of the International Conference on Communications,* June 1995. pp. 1009-1013.