A TRANSFORMATION FOR COMPUTATIONAL LATENCY REDUCTION IN TURBO-MAP DECODING

Arun Raghupathy *

Qualcomm Inc., 6455 Lusk Blvd.,
San Diego, CA-92121

K. J Ray Liu

Electrical Engineering Department
Univ. of Maryland, College Park, MD-20742

ABSTRACT

The SOVA and the log-MAP are commonly used in turbo decoding. In this paper, we propose to modify the sliding window MAP-algorithm in [5] to reduce the computational delay even further. We compare the simulation performance of this low latency log-MAP algorithm with the sliding window log-MAP. We also estimate the VLSI implementation complexities of the SOVA, the log-MAP and the proposed low latency log-MAP.

1. INTRODUCTION

The Soft Output Viterbi Algorithm (SOVA) [2],[3] has been commonly used in turbo decoding. One of the reasons is that this algorithm is relatively less complex when compared with the symbol-by-symbol MAP algorithm. While the original MAP is extremely complex for use in practice, various approximations [4] have been developed such as the Max-Log-MAP and the log-MAP. The log-MAP is equivalent to the MAP but without the implementation problems. The log-MAP uses the max* operation to update the path metrics instead of the max operation [5] that is used in the Max-log-MAP. The log-MAP performs better than the SOVA at low SNR by about 0.5db in AWGN. In fading channels it has been shown [6] that the log-MAP outperforms the normalized SOVA [3] by about 2-3db. The increased complexity of the log-MAP may be justified for fading channels or AWGN channels at very low SNR. A direct implementation of the MAP algorithm requires that we store the path metrics till the end of the block. A sliding window log-MAP algorithm that reduces the path metric storage requirements has been proposed (See for eg. [5]). The computation delay is also reduced using this approach. One of the critical components in iterative decoding is the delay in decoding. The decoding delay is made up of computation delay and interleaver delay [7]. In small frame applications [6], [7] (such as for speech) the computation latency can be a significant fraction of the decoding delay.

In this paper, we propose to modify the algorithm in [5] to reduce the computational delay even further. In serial implementations (such as in [8]), the algorithm modification that we propose also reduces the amount of memory. We compare the simulation performance as well as the VLSI implementation complexities of the SOVA, the log-MAP and our low latency log-MAP algorithm.

*This work was supported in part by the NSF NYI Award MIP9457397 and the ONR grant N00014-93-10566, and was done while this author was at University of Maryland, College Park.

2. LOW-LATENCY MAP ARCHITECTURE

In this section, we develop a low computation delay architecture using fast backward aquisition. In order to motivate our approach, we use the architecture in [5] as a starting point (the architecture corresponding to the method proposed in [5] is summarized below). Fig. 1 shows the evolution in time of the log-MAP algorithm as described in [5]. In Fig. 1, L refers to the acquisition depth (i.e $L \approx 5\nu$, where v is the constraint length). Three processors that operate in parallel are shown in Fig. 1 - the Forward Processor (FP), the Acquisition Backward processor (ABP) and the Backward Processor (BP). The FP generates forward path metrics (FPM's) starting from trellis step k at time $k+2L$. The FPM's are buffered for computing the log likelihood. Starting at time $k+2L$, the ABP traverses through trellis steps $k+2L$ to $k+L$. At time $k+3L$, the ABP produces reliable backward path metrics (BPM). The BP starts with the metrics provided by ABP and produces BPM's corresponding to trellis steps $k+L$ through k. The backward path metrics (BPM's) generated by the BP and the buffered FPM's are combined to compute the log likelihood ratio by a fourth processor (not shown in Fig. 1). Since the log likelihoods are generated in reverse order, a LIFO is used to obtain the results appropriate order. The architecture is shown in Fig. 2(a). Each processor (FP, BP, or ABP) performs an ACS-like operation as shown in Fig. 3. Each ACS unit performs a max* operation instead of the usual max operation in the Viterbi algorithm. The size of the input buffer can be found by looking at the trellis steps $k+L$ to $k+2L$. We notice that the trellis branches from $k+L$ to $k+2L$ are needed until time $k+5L$. This means we need an input buffer size of $4L$ symbols. Similarly, the State Metric Buffer (SMB) should be able to hold the $L \times 2^{\nu}$ PM's that are generated while processing L input symbols.

Our algorithm modification is motivated by the fact that we need the output of the ABP only after L instants (i.e. at the end of the acquisition phase). We are not interested in the intermediate values generated by the ABP. This means that if we are able to process 2 trellis stages at a time (by applying a kind of "look-ahead"), then the acquisition phase of the ABP can be reduced. This is shown in Fig. 1(b) as the "look-ahead" ABP (LA-ABP). The computation latency is reduced to half the previous value. The FPM storage requirements are also reduced as shown in Fig. 2(b). But, in order to process 2 steps at one time the ABP becomes more complicated (as shown in Fig. 4). A larger "look-ahead" factor (greater than 2) will make the imple-

0-7803-5471-0/99/$10.00©1999 IEEE

mentation of the LA-ABP still more complex and therefore practically infeasible. It is obvious that the critical path of the architecture in Fig. 4 is double that of Fig. 3. In order to reduce the critical path of this ACS architecture, it can be shown that we can approximate the computation of $\max^*(a,b,c,d)$ in Fig. 4 with no perceptible loss in performance. The approximate method is shown in Fig. 5 and is explained below (See [1] for details). We can re-write the output in Fig. 4 as

$$\begin{aligned}\max^*(a,b,c,d) &= \max^*(\max^*(a,b),\max^*(c,d)) \\ &= \max(\max^*(a,b),\max^*(c,d)) \\ &\quad +ln(1+exp(-|\max^*(a,b)-\max^*(c,d)|)) \\ &\approx \max(\max^*(a,b),\max^*(c,d)) \\ &\quad +ln(1+exp(-|\max(a,b)-\max(c,d)|))\end{aligned} \tag{1}$$

The first term can be re-written and approximated as

$$\begin{aligned}&\max(\max^*(a,b),\max^*(c,d)) = \\ &\begin{cases}\max^*(a,b) & \text{if } \max^*(a,b) >= \max^*(c,d) \\ \max^*(c,d) & \text{if } \max^*(c,d) > \max^*(a,b)\end{cases} \\ \approx &\begin{cases}\max^*(a,b) & \text{if } \max(a,b) >= \max(c,d) \\ \max^*(c,d) & \text{if } \max(c,d) > \max(a,b)\end{cases}\end{aligned} \tag{2}$$

We refer to the modified algorithm as the approximate low latency log-MAP. The simulation results in Fig. 6 show that the approximation that we applied to simplify the LA-ABP has no perceptible effect on performance. We used the rate 1/2 turbo code in [9] with generators 37,21. Four iterations of turbo decoding were used. An AWGN channel was assumed. An interleaver of size 256 bits (similar to the non-uniform interleaver in [9]) was used in the turbo code. The SOVA results were based on the normalized SOVA (we observed that the un-normalized SOVA was worse than the normalized SOVA by about 0.1-0.2db). The other two curves show the log-MAP and the approximate low latency log-MAP discussed above. Clearly, the log-MAP algorithms perform better than the normalized SOVA by about 0.4db.

3. VLSI DESIGN AND CONCLUSION

For log-MAP algorithms, a parallel state ACS implementation and a parallel likelihood computation leads to two difficulties. The memory bus width between the FP and the SMB is large (since $2^\nu \times L$ PM's are generated by the FP in parallel and have to be written into the SMB). Also, a parallel likelihood computation means that we have to find the $\max^*$ of 2^ν values (which can be very complex). For log-MAP algorithms a serial implementation (as in [8]) is usually more efficient (See [1]). The low latency approximate log-MAP that we have proposed differs from the log-MAP implementation only in the ABP and the size of the input and state memory buffers. The SOVA implementation is similar to [9]. Based on synthesis and layout starting from a Verilog description of the computation units (and estimates of the memory area required for buffers and interleavers/de-interleavers), the results summarized in Table 1 were obtained (See [1] for details). In short, a parallel log-MAP was about 3 times more complex than the SOVA. The parallel low latency log-MAP that we proposed was about 1.5 times as expensive than the log-MAP. The results for serial implementations in Table 1(column 2) show that the serial log-MAP (or the low-latency log-MAP) is about 1.3 times as expensive as the SOVA.

Alg.	$C_{Alg-ITER,p}$ (mm^2)	$C_{Alg-ITER,s}$ (mm^2)
SOVA	22.78	10.45
MAP	68.13	13.18
LL-MAP	100.21	12.56

Table 1: Summary of Area Estimates at 0.8μ for the SOVA, Log-MAP and Low-Latency Log-MAP

We can conclude that improved performance is obtained by using a log-MAP decoder (at the cost of increased complexity). In practice, we need to take into account the nature of the channel (AWGN or fading) while choosing between SOVA and log-MAP. Amongst log-MAP algorithms, if computational delay is a critical parameter (for example in speech applications large delays cannot be tolerated), then the approximate low latency MAP algorithm that we proposed should be used. In particular, when the frame size is small (for eg. 256), our algorithm reduces the overall latency by 12.5% when compared with the algorithm in [5] (the computational latency was reduced from $L \approx 30$ in [5] to $L/2$) with almost no cost in terms of BER (See [1] for details).

4. REFERENCES

[1] A. Raghupathy, "Low Power and High Speed Algorithms and VLSI Archite ctures for Error Control Coding and Adaptive Video Scaling," *Ph.D Thesis*, University of Maryland, College Park, Dec. 1998.

[2] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. on Info. Theory*, vol. 42, pp. 429–445, Mar. 1996.

[3] L. Papke, P. Robertson, and E. Villebrun, "Improved decoding with the SOVA in a parallel concatenated (turbo-code) scheme," in *IEEE Int. Conf. on Communications*, pp. 102–106, June 1996.

[4] P. Robertson, P. Hoeher, and E. Villebrun, "Optimal and sub-optimal maximum a posteriori algorithms suitable for turbo decoding," *European Trans. on Telecomm.*, vol. 8, pp. 119–125, Mar.-Apr. 1997.

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

[6] M. Valenti and B. Woerner, "Performance of turbo codes in interleaved flat fading channels with estimated channel state information," in *IEEE Vehicular Tech. Conf.*, pp. 66–70, IEEE, May 1998.

[7] P. Jung, "Comparison of turbo-code decoders applied to short frame transmission systems," *IEEE J. Selected Areas in Communications*, vol. 14, pp. 530–537, Apr. 1996.

[8] S. S. Pietrobon, "Implementation and performance of a turbo/MAP decoder," *Intl. J. of Satellite Communications*, vol. 16, pp. 23–46, Jan.-Feb. 1998.

[9] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: Turbo codes," *IEEE Trans. Comm.*, vol. 44, pp. 1261–1271, Oct. 1996.

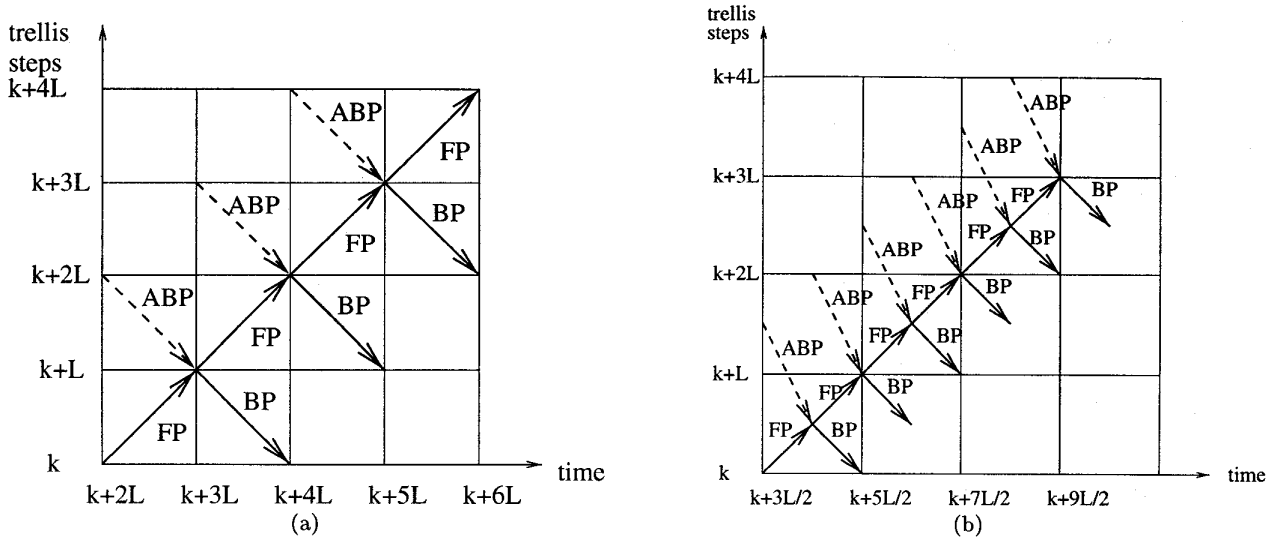

Figure 1: (a) Evolution in time of the original log-MAP algorithm (b) Evolution in time of the modified log-MAP algorithm

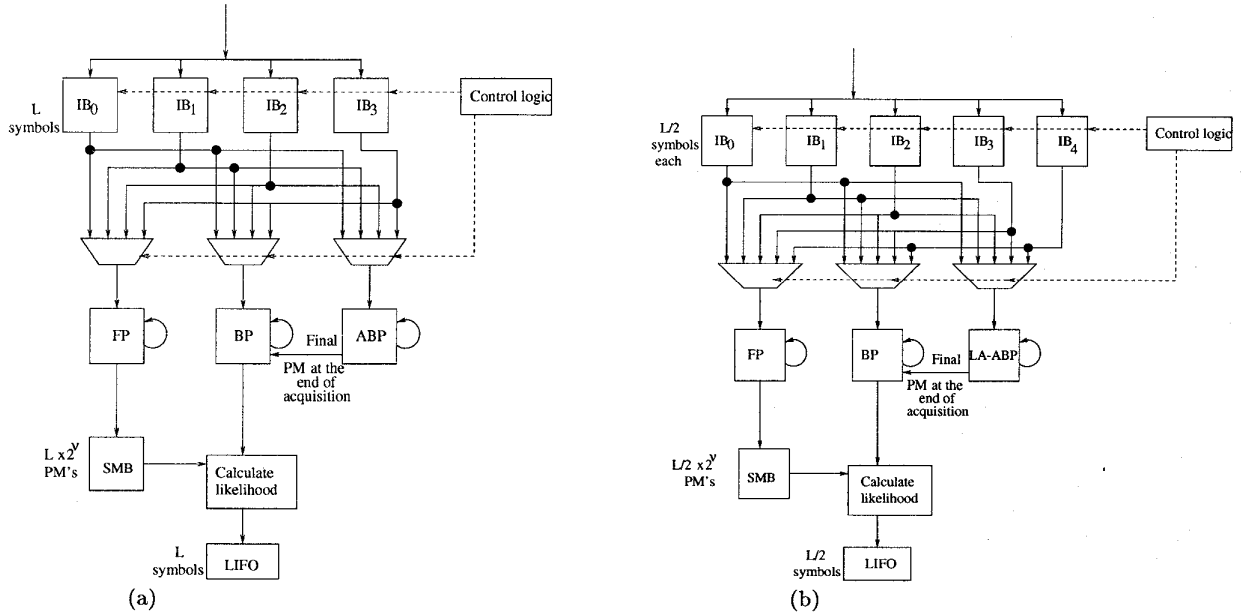

Figure 2: (a) Overall architecture of log-MAP (b) Architecture using acquisition look ahead

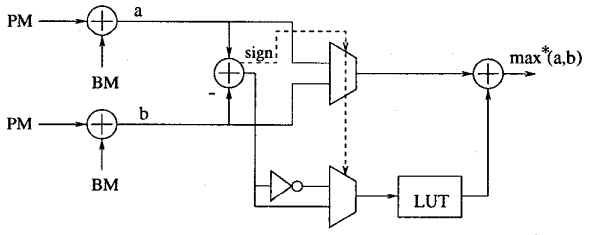

Figure 3: ACS architecture for log-MAP

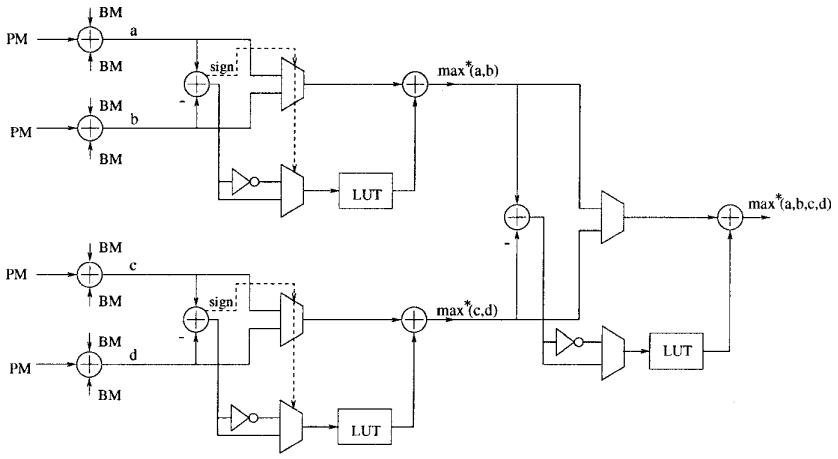

Figure 4: ACS architecture for acquisition backward processor in modified log-MAP

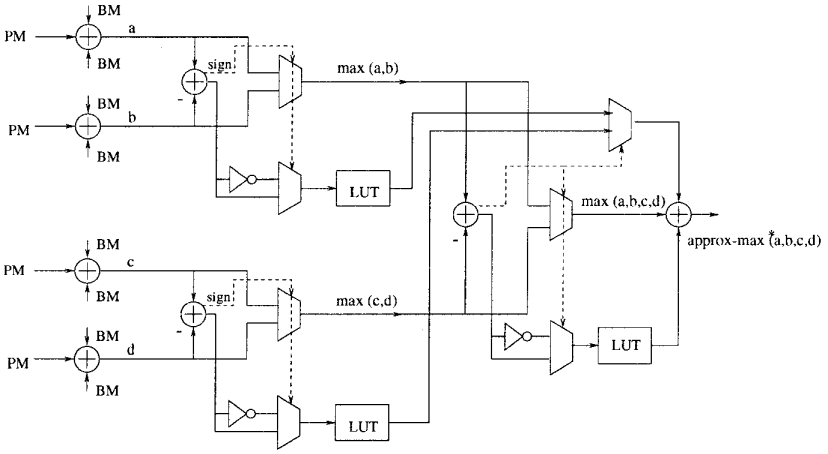

Figure 5: Approximate ACS architecture for acquisition backward processor in modified log-MAP

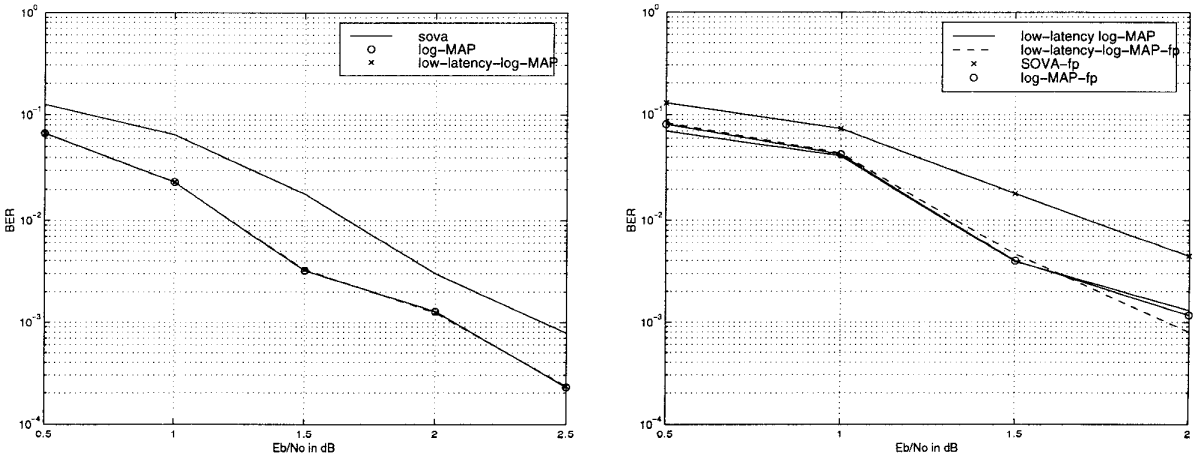

Figure 6: (a) Comparative performance of SOVA, log-MAP and modified log-MAP (b) Fixed Point Performance