

Concurrent Turbo-Decoding

Brendan J. Frey (frey@cs.utoronto.ca)¹, Frank R. Kschischang, and P. Glenn Gulak
Dept. of Electrical and Computer Engineering, University of Toronto, Toronto, Canada M5S 3G4

Abstract — The turbo-decoding algorithm can be viewed as a message passing procedure on a graph. We contrast the standard “forward-backward” turbo-decoding algorithm with a new “concurrent” algorithm that is suited to parallel implementation.

I. INTRODUCTION

Many iterative decoding algorithms can be viewed as a message passing procedure on a graph, where each vertex is associated with an information symbol, a state variable, a codeword symbol, or a received signal [1, 2]. When a vertex is observed (e.g., a channel output demodulated), a *message* is sent out on each of its edges. When a vertex receives a message on an edge, it modifies its own status and then sends out a new message on each other edge. Messages need not be sent out immediately; they can be buffered and sent out at any time. These concepts lead naturally to the idea of a message passing *schedule*, which specifies when and where to pass messages.

In cycle-free graphs, this message passing procedure computes the marginal probability for each variable given the observations *exactly*, regardless of the schedule used [3]. For example, the forward-backward (BCJR) decoding algorithm is a special case of this procedure applied to a convolutional code chain to compute $P(\text{symbol } k | \text{channel output})$.

II. STANDARD AND CONCURRENT TURBO-DECODING

The graphs for several interesting codes (such as turbo-codes) contain cycles and in these cases, the decoder message passing schedule *does* affect the final BER. Standard turbo-decoding makes use of a *forward-backward* schedule [4], where $\alpha\beta$ -messages are passed along the chain for one convolutional code, first in the forward direction and then in the backward direction. *Extrinsic information messages* are then passed through an interleaver to the other chain, which is also processed in the forward-backward manner. For reasonably large encoder memories (say, $\nu \geq 3$), the time spent computing the $\alpha\beta$ -messages dominates the time spent computing the extrinsic information messages.

One time step of *concurrent turbo-decoding* consists of simultaneously passing messages in both directions on all graph edges. Notice that concurrent turbo-decoding is not just a parallel implementation of standard turbo-decoding. It is a different algorithm which may have different properties. A detailed description of this algorithm can be found in [2]. For reasonably long block lengths, a prohibitively large number of simple processors would be needed for a fully parallel VLSI implementation of concurrent turbo-decoding. In this paper, we empirically compare the time complexity of standard decoding with concurrent decoding, while ignoring practical implementation issues such as space complexity. These results suggest that efficient implementations (e.g., time-shared processors) may lead to faster decoding

¹This work was supported by funding from NSERC and ITRC.

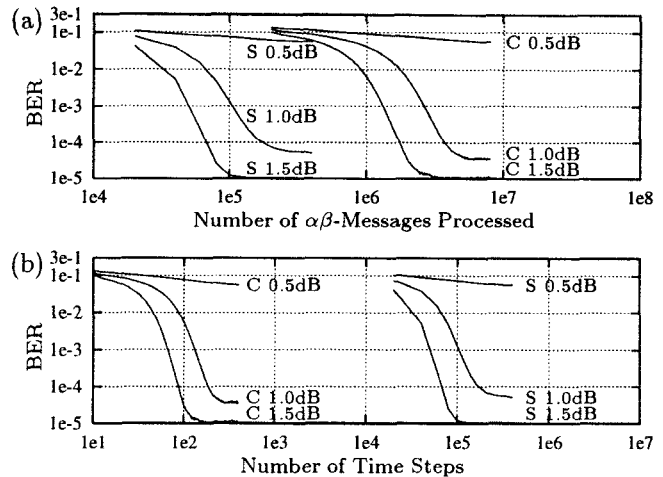


Figure 1: Performance of standard (S) and concurrent (C) turbo-decoding when implemented on a serial computer (a) and on a parallel computer (b), for 3 values of E_b/N_0 .

III. RESULTS

For each of three AWGN channels, we simulated the transmission of 10^7 information bits for a rate 1/2, 5000-information bit, $2 \times (21_s/37_s)$, punctured turbo-code with a randomly selected permutter. Figure 1a shows BER versus number of $\alpha\beta$ -messages processed, for standard and concurrent turbo-decoding. The number of $\alpha\beta$ -messages gives a good indication of decoding complexity on a serial computer. The standard algorithm is better suited to serial implementation. Figure 1b shows BER versus number of time steps for the case where 10,000 processors are available for concurrent turbo-decoding. We assume that pipelining is *not* used for standard turbo-decoding. For $E_b/N_0 = 1.5$ dB, concurrent turbo-decoding is 850 times faster at a BER of 10^{-5} . If one processor is used for each half iteration of 5 iterations of standard turbo-decoding in a pipeline fashion, standard decoding can be sped up by a factor of only 10 (extra pipeline stages do not improve the BER). Concurrent turbo-decoding is still 85 times faster.

We thank D. J. C. MacKay and R. M. Neal for discussions.

REFERENCES

- [1] B. J. Frey and F. R. Kschischang, “Probability propagation and iterative decoding,” in *Proceedings of the 34th Allerton Conference*, 1996. <ftp://ftp.cs.utoronto.ca/pub/frey/pp-allerton.ps.Z>
- [2] B. J. Frey, *Bayesian Networks for Pattern Classification, Data Compression and Channel Coding*. Ph.D. Thesis, Dept. of Electrical and Computer Eng., University of Toronto, 1997. <ftp://ftp.cs.utoronto.ca/pub/frey/thesis.ps.Z>
- [3] J. Pearl, *Probabilistic Reasoning in Intelligent Systems*. San Mateo CA.: Morgan Kaufmann, 1988.
- [4] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: Turbo codes,” in *Proceedings of the IEEE International Conference on Communications*, 1993.