A HIGH THROUGHPUT LIMITED SEARCH TRELLIS DECODER FOR CONVOLUTIONAL CODE DECODING

Tong Zhang

Electrical, Computer and Systems Engineering Department
Rensselaer Polytechnic Institute
Troy, New York, USA
tzhang@ecse.rpi.edu

ABSTRACT

Due to the lack of operational parallelism and structured data storage/retrieval, limited search trellis decoding algorithms have been traditionally ruled out for applications demanding high throughput convolutional code decoding. Among various limited search algorithms, the *T*-algorithm performs breadth-first limited search and has good potential for parallel decoding. In this paper, we propose two techniques at the algorithm and VLSI architecture levels for the T-algorithm to improve the decoding parallelism and tackle the data storage/retrieval problem, which enables the high throughput path-parallel *T*-algorithm decoder VLSI implementation. This work provides a vehicle for exploiting the merits of the *T*-algorithm, i.e., low computational complexity that is adaptive to the channel distortion, in high throughput applications.

1. INTRODUCTION

Convolutional code with Viterbi decoding is widely used in digital communication systems. Viterbi algorithm (VA) performs breadth-first exhaustive search on the code trellis to achieve the maximum likelihood (ML) decoding at the cost of a computational complexity increasing exponentially with the memory order of the convolutional code encoder. In contrast, limited search decoding algorithms [1] perform non-exhaustive search on the code trellis to achieve or approximate the ML decoding. The computational complexity of limited search algorithms is generally much less than that of VA. Moreover, many limited search decoding algorithms exhibit inherent adaptation of the computational effort to the channel distortion. This *effort-self-adaptation* together with the low computational complexity show great promise to realize low complexity, low power convolutional code decoders.

However, the real-world application of limited search decoding algorithms lags far behind of that of VA. This is mainly due to the fact that most limited search decoding algorithms, in their current formulations, lack the operational parallelism and structured data storage/retrieval, which makes their high throughput parallel VLSI decoder implementations problematic. As breadth-first limited search algorithms, *M*-algorithm [2] and *T*-algorithm [3] have great potential for high throughput path-parallel decoding. Nevertheless, all the previous work only targeted on the **path-serial** hardware decoder design of these two algorithms. Bengough and Simons [4] and Simons [5, 6] developed several sorting-based and nonsorting-based path-serial *M*-/*T*- algorithm decoders. Chan et al. [7] developed a path-serial modified *T*-algorithm decoder that is similar to the state-serial Viterbi decoder. We note that the drawback of unstructured data storage/retrieval in *M*-/*T*- algorithms is *concealed* by the path-serial decoding process and thus is not a critical issue in path-serial decoders. The power efficiency of *T*-algorithm over VA has been demonstrated by computer simulations [8].

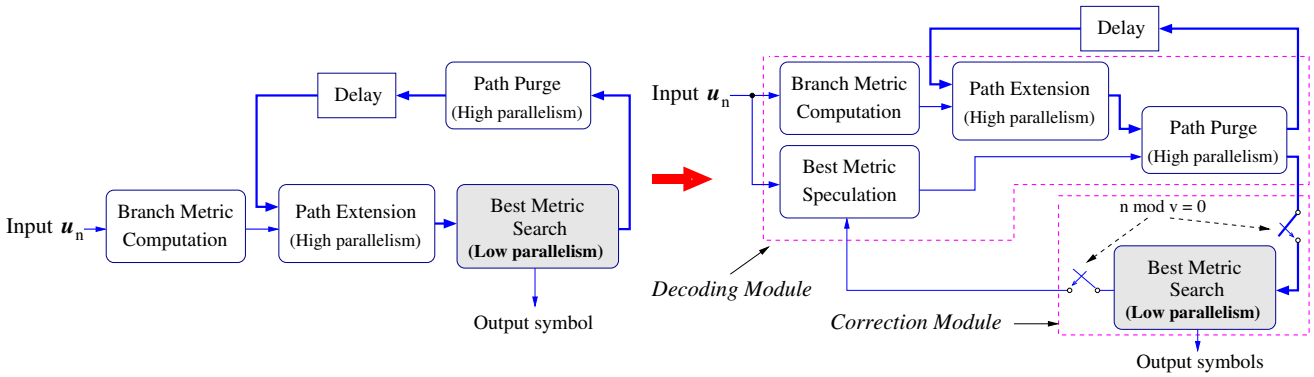
Distinguished from previous work, in this paper, we present a high throughput **path-parallel** *T*-algorithm decoder design. We first propose a speculation technique to modify the original *T*-algorithm, leading to a SPEC-*T* algorithm, to speed up the parallel decoding. The modified *T*-algorithm in [7] can be considered as a special case of SPEC-*T* algorithm when the speedup is minimized. In sharp contrast to the path-serial decoder design, the unstructured data storage/retrieval becomes a big challenge in the path-parallel decoder design. To tackle this problem, we develop a data re-distribution scheme based on token bus to support parallel data storage/retrieval. Compared with a **state-parallel** Viterbi decoder, to achieve comparable error-correcting performance, this path-parallel SPEC-*T* decoder can achieve only few times slower throughput while consuming much (even an order of magnitude) less energy consumption and silicon area.

2. BACKGROUND

Broadly speaking, breadth-first decoding algorithms extend all the survivor paths at each trellis depth at once, purge some paths according to certain criterion, and then continue on to the next trellis depth. Various breadth-first algorithms primarily differ on the purging rules. Readers are referred to [1, 2] for detailed analysis of different breadth-first algorithms.

In *T*-algorithm, at each decoding depth, all the paths whose cumulative path metric falls outside of a *retention band* will be purged. Its operations at each depth are as follows:

1. *Branch Metric Computation & Path Extension*: Given input data $\mathbf{u}_n$ at depth n, compute the branch metrics and extend the survivor paths from the previous depth to obtain the contender paths.
2. *Best Metric Search*: Find the contender path having the best (largest) path metric, denoted as $\Gamma_B^{(n)}$, and release its oldest path symbol as the output symbol.
3. *Path Purge*: (a) Purge the contender path whose metric $\Gamma^{(n)}$ satisfies $\Gamma_B^{(n)} - \Gamma^{(n)} > T$, where *T* is a pre-specified



(a) The original T−algorithm

(b) The high throughput SPEC−T algorithm

Fig. 1. The structure of original *T*-algorithm and SPEC-*T* algorithm.

threshold, and (b) purge all the contender paths that do not agree with the output symbol.

T-algorithm can achieve near-ML decoding performance with the average number of survivors much less than the number of trellis states. However, it is a challenge to implement a high throughput path-parallel *T*-algorithm VLSI decoder mainly due to the following issues:

Algorithm-inherent path-parallel decoding throughput bottleneck: As remarked in Fig. 1 (a), although we can perform the *Path Extension* and *Path Purge* in parallel among all the paths with a delay of only few additions and subtractions, the *Best Metric Search* incurs a relatively large delay due to the **serial** essence of the search operation, which prevents the path-parallel decoder from achieving high throughput.

Unstructured data storage/retrieval: To enable path-parallel decoding, the decoder should be able to read and update all the survivor path data in parallel. However, the set of survivor paths varies from each decoding depth to the next. This makes the parallel path data access pattern *dynamic* and *unstructured*, whereas VLSI implementations always favor static and structured parallel data access patterns such as that of state-parallel VA decoder. The unstructured parallel data storage/retrieval may significantly degrade the throughput and power consumption performance of path-parallel decoder.

3. PROPOSED SPEC-*T* ALGORITHM

In this section, we present a modified *T*-algorithm, called SPEC-*T* algorithm, in which the search-the-best-metric throughput bottleneck is eliminated. The basic idea is *best metric speculation with lagged correction*: instead of searching the exact best metric at each depth, we *speculate* the best metric based on the current input and perform an *off-the-main-recursion* search to correct the speculation error with a certain delay. Fig. 1 (b) shows the data-flow diagram of the SPEC-*T* algorithm, which contains two recursive data-paths performed by two modules, respectively:

Decoding Module performs the branch metric calculation, path extension, and path purge by *speculating* the best metric. Given a fixed integer v as one decoder parameter, at depth n, it speculates the best metric $\hat{\Gamma}_B^{(n)}$ as follows:

- If $n \mod v \neq 0$, then $\hat{\Gamma}_B^{(n)} = \hat{\Gamma}_B^{(n-1)} + BM_B^{(n)}$, where $BM_B^{(n)}$ is the best branch metric given the input data $\mathbf{u}_n$. Notice that we may directly obtain $BM_B^{(n)}$ from the branch that matches the hard decision of input $\mathbf{u}_n$.
- If $n \mod v = 0$, then $\hat{\Gamma}_B^{(n)} = \hat{\Gamma}_B^{(n-1)} + BM_B^{(n)} - E$, where E is provided by the Correction Module to compensate for the accumulated speculation error.

When $n \mod v = 0$, the Decoding Module passes $\{\hat{\Gamma}_B^{(n)} - \Gamma_i^{(n)}\}$ and the v oldest path symbols of all the contender paths to the Correction Module.

Correction Module performs a *search-the-best-metric* operation to adjust the speculated best metric. It searches for $E = \min\{\hat{\Gamma}_B^{(n)} - \Gamma_i^{(n)}\}$ and releases the v symbols associated with the best path as the decoding output. When $n \mod v = 0$, it sends E to the Decoding Module and receives a new set of $\{\hat{\Gamma}_B^{(n)} - \Gamma_i^{(n)}\}$ and path symbols from the Decoding Module.

It is clear that the Decoding Module can operate on all the paths in fully parallel without any serial search operation, and hence can realize high throughput path-parallel decoding. In order to compenstate the unmatched delay of these two modules, Correction Module runs v times slower than Decoding Module. The value of v is determined by the specific implementations. We note that, when $v = 1$, SPEC-*T* algorithm reduces to the modified *T*-algorithm proposed in [7].

The *lagged* correction of the speculation error inevitably results in certain performance degradation, particularly when v is large. The question is whether such degradation is tolerable for the specific applications. As we will show later in one example, the performance degradation is very small.

4. PATH-PARALLEL SPEC-*T* DECODER DESIGN

Fig. 2 shows the proposed path-parallel SPEC-*T* decoder structure, which uses a data re-distribution scheme based on a *token bus* to tackle the unstructured data storage/retrieval problem. Let

M denote the maximum number of survivors kept at each decoding depth, which is typically much less than the number of trellis states. This decoder contains M processing element PEs. Each PE_i extends one survivor path and performs the path purge, where the corresponding survivor path data is stored in register array PD_i. To enable path-parallel decoding, each PD_i should contain at most one survivor path at the beginning of each decoding depth. However, at the end of each decoding depth, some PDs may contain more than one survivor path and some others may not contain any survivor paths. This demands *re-distribute* path data among PDs at the end of each decoding depth to make each PD_i contain at most one survivor path.

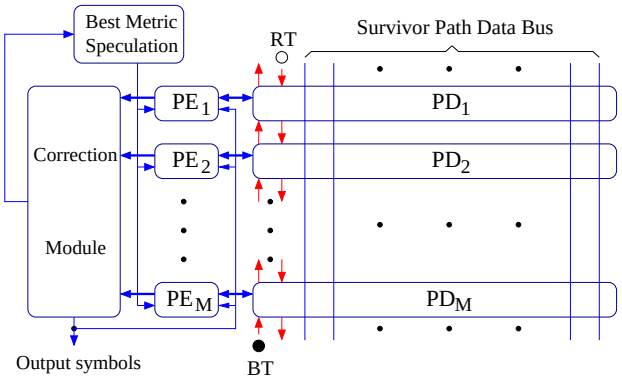


Fig. 2. The path-parallel decoder structure.

We develop a scheme using the concept of token bus, as illustrated in Fig. 2, to effectively realize such data re-distribution. Notice that after the path purge, based on their contents, the PDs fall into three categories: (1) empty register arrays PD_i^E that do not contain any survivor paths, (2) carefree register arrays PD_i^C that only contain one survivor path, and (3) congested register arrays PD_i^G that contain more than one survivor path. As shown in Fig. 2, after the path purge, the decoder launches two tokens, *broadcasting* token BT and *receiving* token RT, which flow through the PDs. We have the following operation rules:

1. Carefree register array PD_i^C simply bypasses both the BT and RT;
2. Empty register array PD_i^E intercepts the RT and bypasses the BT;
3. Congested register array PD_i^G intercepts the BT and bypasses the RT.

The PD_i^G that holds BT broadcasts one survivor path data to the data bus at once, which is received by the PD_i^C that holds RT. A PD_i^G that contains g survivor paths releases the BT after it has broadcasted $g-1$ survivor path data to the bus. The PD_i^E will release the RT once it receives one survivor path data. The data broadcast-receive operation will terminate whenever one token, BT or RT, reaches the other end. If RT is the first, the decoder knows that the number of survivor paths exceeds the limit M, which is called decoding overflow, and repeats the current decoding depth by reducing the speculated best metric $\hat{\Gamma}_B^{(k)}$ by a pre-specified value R.

We note that this path-parallel SPEC-*T* decoder has four desirable properties from the VLSI implmenetation perspective:

1. The decoder has a regular structure with low interconnection complexity.
2. Because the average number of survivors is small as shown later in one example, it is reasonable to expect that the power consumption due to the computation and data re-distribution is low.
3. Because of the simple operations involved in the data re-distribution, the clock that controls the data re-distribution can run much faster than the clock that controls the operation of PEs. Thus the overall delay incurred by the data re-distribution will not be significant.
4. The decoder can detect the occurrence of overflow by simply observing the arrival of the BT and RT.

5. SIMULATION RESULTS

Consider the decoding of 1024 information bits per frame, rate-1/2 convolutional codes modulated by BPSK and transmitted over an AWGN channel. For both *T*-algorithm and SPEC-*T* algorithm, we use ODP (Optimum Distance Profile) systematic feed-forward convolutional encoder, as suggested in [10]. The generator $G_s = (2000000, 7144761)$ (in octal notation) has the memory of $m = 19$. For VA, we use three encoder generators: $G_v^{(4)} = (35, 23)$ with $m = 4$, $G_v^{(6)} = (171, 133)$ with $m = 6$, and $G_v^{(8)} = (753, 561)$ with $m = 8$ (used in IS-95 standard).

With the transmission power of each codeword bit normalized as 1, we use the following decoder configurations: for the original *T*-algorithm, $T = 6$; for SPEC-*T* algorithm, $T = 8$, $v = 4$, and $R = 1$. Fig. 3 shows the simulated BER vs. SNR and the average number of survivors at each decoding depth. Notice that the numbers of survivors are fixed as $2^4 = 16$, $2^6 = 64$, and $2^8 = 256$ in Viterbi decoders. We note that, compared with *T*-algorithm, the performance degradation of SPEC-*T* due to the lagged (by 4 depths in this example) best metric correction of the speculation error is not significant.

In Table 5, we present the simulation results of two parameters pertaining to the path-parallel SPEC-*T* decoder throughput:

1. the average number of overflows in each 1024-bit frame, denoted as DL_o, and
2. the average number of the data broadcast-receive operations to complete the data re-distribution at each decoding depth, denoted as NC_r.

SNR		2 dB	3 dB	4 dB	5 dB	6 dB
$M = 64$	DL_o	50.5	24.0	11.5	5.0	1.8
	NC_r	6.8	5.4	4.6	4.1	3.7
$M = 128$	DL_o	20.0	7.0	2.0	0.6	0.1
	NC_r	7.9	5.9	4.8	4.1	3.7

Table 1. Simulation results of two parameters pertaining to the decoder throughput.

6. CONCLUSIONS

This paper presents two techniques at the algorithm and VLSI architecture levels for hardware implementation of path-parallel

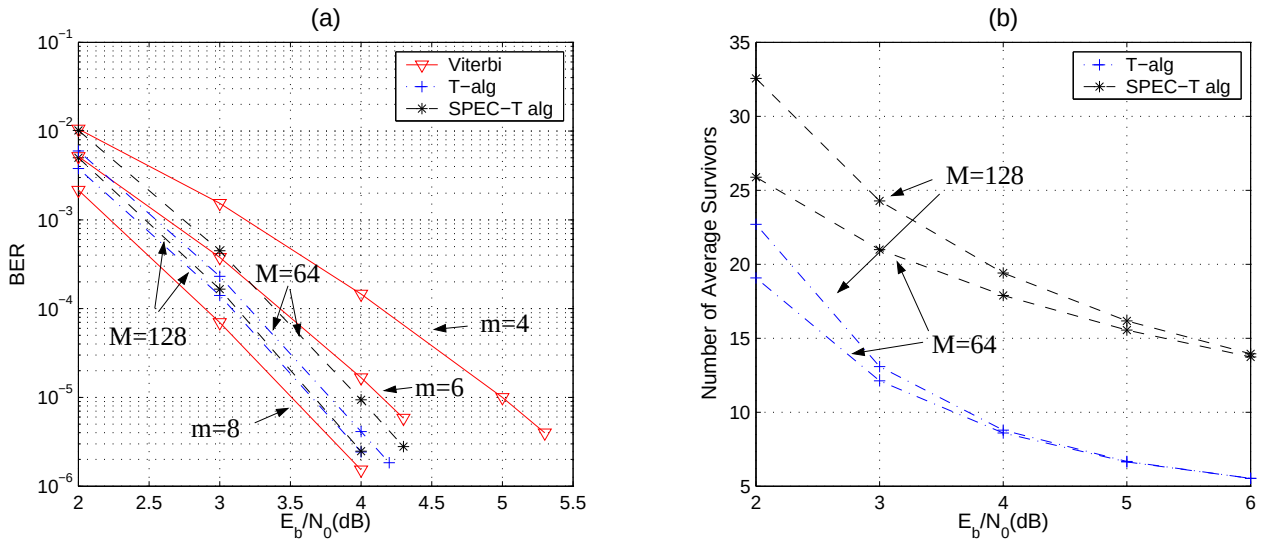


Fig. 3. Simulation results of (a) BER vs. SNR, and (b) average number of survivors vs. SNR.

high throughput *T*-algorithm decoder. A SPEC-*T* algorithm is proposed to eliminate the algorithm-inherent search-the-best-metric decoding throughput bottleneck in the original *T*-algorithm. We develop a path-parallel SPEC-*T* decoder structure, in which a data re-distribution scheme based on token bus is proposed to tackle the design challenge due to the unstructured data storage/retrieval. Such path-parallel high throughput SPEC-*T* decoder provides an unique opportunity to exploit the attributes of the *T*-algorithm, i.e., low computational complexity that is adaptive to the channel distortion, to significantly reduce the convolutional code decoder power consumption and implementation complexity while achieving very high decoding throughput.

7. REFERENCES

[1] J. B. Anderson, "Limited search trellis decoding of convolutional codes," *IEEE Transactions on Information Theory*, vol. 35, pp. 944–955, Sept. 1989.

[2] J. B. Anderson and S. Mohan, "Sequential coding algorithms: A survey and cost analysis," *IEEE Transactions on Communications*, vol. 32, pp. 169–176, Feb. 1984.

[3] S. J. Simmons, "Breadth-first trellis decoding with adaptive effort," *IEEE Transactions on Communications*, vol. 38, pp. 3–12, Jan. 1990.

[4] P. A. Bengough and S. J. Simmons, "Sorting-based VLSI architectures for the M-algorithm and T-algorithm trellis decoders," *IEEE Transactions on Communications*, vol. 43, pp. 514–522, Feb. 1995.

[5] S. J. Simmons, "A nonsorting VLSI structure for implementing the (M, L) algorithm," *IEEE Journal on Selected Areas in Communications*, vol. 6, pp. 538–546, April 1988.

[6] S. J. Simmons, "A bitonic-sorter based VLSI implementation of the M-algorithm," in *Proc. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, June 1989, pp. 337–340.

[7] M.-H. Chan, W.-T Lee, M.-C. Lin, and L.-G. Chen, "IC design of an adaptive Viterbi decoder," *IEEE Transactions on Consumer Electronics*, vol. 42, pp. 52–62, Feb. 1996.

[8] R. Henning and C. Chakrabarti, "Low-power approach for decoding convolutional codes with adaptive viterbi algorithm approximations," in *Proceedings of the 2002 International Low Power Electronics and Design*, 2002, pp. 68–71.

[9] G. J. Pottie, "Low latency sequential decoding," in *Proc. IEEE International Symposium on Information Theory*, July 1997, p. 499.

[10] H. Osthoff, J. B. Anderson, R. Johannesson, and C.-F. Lin, "Systematic feed-forward convolutional encoders are better than other encoders with an M-algorithm decoder," *IEEE Transactions on Information Theory*, vol. 44, pp. 831–838, March 1998.