A Parallel Viterbi Decoder Implementation for High Throughput

Muhammad Shoaib Bin Altaf
shoaibbinalt@wisc.edu

ABSTRACT

Today's data reconstruction in digital communication systems requires designs of highest throughput rate. The Viterbi Algorithm is a key element in such digital signal processing applications. The non-linear and recursive nature of the Viterbi decoder makes its high-speed implementation challenging. Several promising approaches to achieve high throughput have been proposed in the past. In this project one such technique, look-ahead, is studied for extracting vectorized output bits without taking into consideration the hardware cost involved. It came out that even with small steps of looking ahead the processing gains are very high.

1. INTRODUCTION

Convolutional coding and Viterbi decoding are widely used in modern digital communication systems, such as communication, satellite communication, and mobile communication, to achieve low-error rate data transmission. The Viterbi algorithm [10] is known to be an efficient method for the realization of maximum likelihood (ML) decoding of convolutional codes. It is based on the study of a weighted graph that is used for attempting the reconstruction of the input sequence to the convolutinal encoder based on the coded sequence received from a noisy channel.

With the development of digital communications, high speed large Viterbi decoder are required to yield higher coding gain and provide large ability to transmit more data in the same channel. Latest wireless communication technologies like WiMAX is opening up new challenges in baseband hardware design and demanding high speed, area efficient and reconfigurable designs. As the data speeds are moving skywards, demand for high speed Viterbi decoders is increasing. The proposed design particularly will strive for a design optimized for high speed such that the at each clock cycle the decoder will be decoding a couple of bits instead of a single bit without compromising on performance so that it suits the requirements of latest wireless standards like 802.16e.

The add-compare select unit recursion in the Viterbi decoding algorithm contains feedback loops the speed of Viterbi decoding is limited by this iteration bound. In this project, I have implemented a look-ahead technique for combining several trellis steps into one trellis step in time sequence, for breaking the iteration bound of the Viterbi decoding algorithm. Results show that the proposed design gave a considerable gain in processing time. The next section contains the previous work done. Section 3 describes the actual Algorithm followed by section 4 containing information about the optimizations techniques. Section 5 describes the implementations followed by results in section 6 and conclusions in section 7.

2. RELATED WORK

Researchers have been working in the area of optimizing the Viterbi decoder for couple of years. Some tried to increase the throughput & reduce the area consumed by giving the idea of radix-4 approach instead of radix-2 [17]. The idea of systolic array processing is also not new in this regard for increasing the throughput [15]. Nandu et al. employed normalization & systolic array processing in their design [7]. Batcha et al. achieved high throughputs compatible with Wimax using pipelining [6]. Fettweis [1] also used pipelining for increasing the throughput of their design. The idea of using look-ahead for speeding up sequential architectures is also not new, Parhi investigated look-ahead for Huffman decoding in the early 90's [5].

3. VITERBI ALGORITHM

We can view the Viterbi algorithm as a dynamic programming algorithm for finding the shortest path through a trellis, and the algorithm can be broken down into the following three steps.

- Weigh the trellis; that is, calculate the branch metrics.
- Recursively compute the shortest paths to time *n*, in terms of the shortest paths to time *n-1*. In this step, decisions are used to recursively update the survivor path of the signal. This is known as add-compare-select (ACS) recursion.
- Recursively find the shortest path leading to each trellis state using the decisions from Step 2. The shortest path is called the survivor path for that state and the process is referred to as survivor path decode. Finally, if all survivor paths are traced back in time, they merge into a unique path, which is the most likely signal path that we are trying to find.

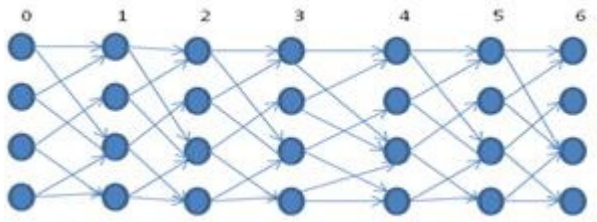

Figure 1- Actual Flow of Viterbi Algorithm

Associated with each trellis state *S* at time *n* is a state metric which is the accumulated metric along the shortest path leading to that state. The state metrics at time *n* can be recursively calculated in terms of the state metrics of the previous iteration as follows:

$$\mathbf{PM_{i+1} = min\ (PM_i + BM_{i,i+1},\ PM_j + BM_{j,i+1});} \quad (1)$$

$$\mathbf{PM_{j+1} = min\ (PM_i + BM_{i,j+1},\ PM_j + BM_{j,j+1});} \quad (2)$$

where *i+1* is a predecessor state of *i* and $BM_{i,i+1}$ is the branch metric on the transition from state *i* to state *j*. The qualitative interpretation of this expression is as follows. By definition, the shortest path into state *j* must pass through a predecessor state. If

the shortest path into *j* passes through *i*, then the state metric for the path must be given by the state metric for *i* plus the branch metric for the state transition from *i* to *j*. The final state metric for *j* is given by the minimum of all possible paths. The recursive update described by (1) and (2) are the ACS operation and are implemented as shown in Fig. 1 for the four-state trellis example.

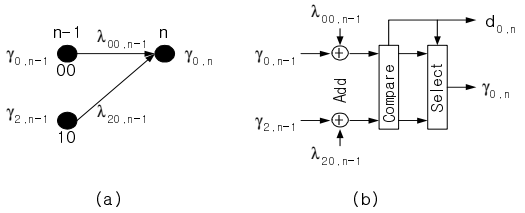

Figure 2 (a) Predecessor states of state 00 (b) State metric update for state 00, implemented using 2-way ACS

The update unit is referred to as a two-way ACS unit, because there are two input branches for each state, In general, a state with m-input branches requires an m-way ACS unit[2, 7]. As well as calculating the updated state metric, the ACS unit outputs a decision $d_{s,n}$, which identifies the entering path of the minimum metric.

In order that the input sequence can be decoded, the survivor path (shortest path) or signal through the trellis must be traced and decoded. The two classical algorithms for survivor path storage and decoding are the register-exchange method and the trace-back method. Both algorithms require a recursive update which fundamentally limits the throughput. Register-exchange is suitable for low-complexity trellises and is high in throughput. Trace-back is preferable for higher complexity trellises due to reduced area and power dissipation, In the Viterbi decoder, the register-exchange method is used to finish the survivor path storage and decoding.

Since traceback approach lags behind Register exchange in throughput, it's more logical to apply throughput increasing techniques on the traceback approach rather than register exchange. With this methodology, I will be getting the reduced area but also higher throughput.

4. OPTIMIZATIONS

A high speed implementation of the VA can only be achieved by increasing the speed of computation of all its three units. Since the ACS unit is much more complex, it is bottleneck which limits the throughput rate but as every cloud has a silver lining, the "Good" thing is that VA is recursive in nature. I have number of loops in the algorithm. So more or less all of the optimization techniques we studies in the course can be successfully employed on VA to have a better performance. Some of the possible candidates can be

- Look Ahead Transformation
- Loop unrolling
- Retiming
- Systolic Array implementation
- Pipelining

As mentioned in Section 2, lot of work has been done using above techniques mentioned above by exploiting the recursive nature of the Viterbi Algorithm.

Since the main purpose of this project is to come up with such a technique which gives a vectorized output, the only viable option is "Look-Ahead Transformation". Other techniques do help in increasing throughput or in general performance but they don't serve the very purpose of giving notion parallel decoding of number of bits. So for the rest of section, I will discuss the implementation optimization using Look-Ahead technique only.

4.1Look-Ahead Transformation

The difficulty of pipelining the feedback algorithm was removed by the use of look-ahead computation. Look-ahead can be used in the form of pipelining [2], parallel processing [3], or both. In order to understand the process of look-ahead, consider the flow in the normal executaion in Figure 1. At each time interval, branch metrics will be computed and the best path metric will be decided on the basis of minimla vlue of the two possible path. This process will continue till the trellis is completetly fileld. The number of hops taken will be equal to the number of time stamps. Figur 2 depicts this that for filling a trellis to '6' time units, 6 compariosn needed to be done in '6' hops.

Now consider Figure 3, the actual flow is modified in such a way that instead of going to time1 it goes directly to time2, this will save the time spend at each node. In the new scenerio the

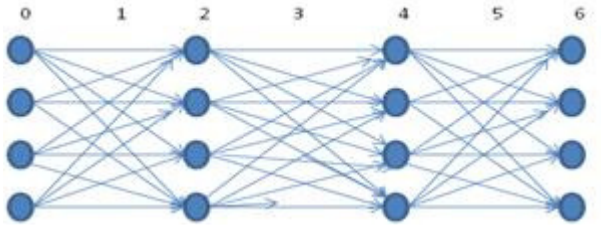

Figure 3- Look-Ahead Transformation M=1

compariosn will be done skipping each intermediate node. It is also clear that earlier there were only two paths to reach a node but with this modification the new path can enter a node from any of the four nodes. This will incrase the size of my comparator at each node, instead of selecting the best (minimum) between two, now it has to slect the minimum among the four.

Similarly, taking the look-ahead step M=2, I will skip 2 intermediate nodes and ecah time jump to every 3rd node after adding the respective branch metrics with the state metric. This idea can be extended to skip all the intermediate stages & go directly to the final node. But it depends on our application what we are looking for.

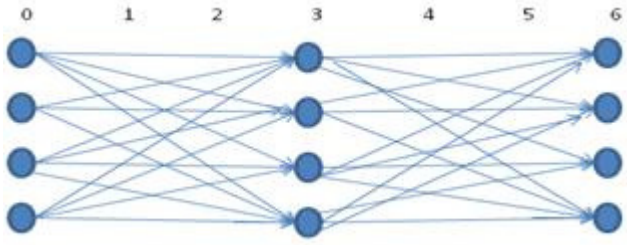

Figure 4- Look-Ahead Transformation M=2

The expressions in (1) and (2) now become

$$PM_{i+1} = \min(PM_i+BM_{i,i+1}, PM_j+BM_{j,i+1}, PM_k+BM_{k,i+1}, PM_l+BM_{l,i+1}); \quad (3)$$

$$PM_{j+1} = \min(PM_i+BM_{i,j+1}, PM_j+BM_{j,j+1}, PM_k+BM_{k,j+1}, PM_l+BM_{l,j+1}); \quad (4)$$

$$PM_{k+1} = \min(PM_i+BM_{i,k+1}, PM_j+BM_{j,k+1}, PM_k+BM_{k,k+1}, PM_l+BM_{l,k+1}); \quad (5)$$

$$PM_{l+1} = \min(PM_i+BM_{i,l+1}, PM_j+BM_{j,l+1}, PM_k+BM_{k,l+1}, PM_l+BM_{l,l+1}); \quad (6)$$

These effects are also visble in Figure1-3.
After completion of filling the trellis the next step is decoding the bits from the information gethered. Genrally we start from the end and move each time interval back to get to the adjacent previous best state. Thus if there are '6' bits in my block I have to go back 6 hops decoding a single bit during each backward hop. Also I know that the present state can be achieved form two possibel states the decoding table will be simple as well.
But with the new implementation, say M=1 I will skip '5' will directly go to '4' and judge from the metrcs computed that which among the four is the best previous state. Thus increasing the complexity of the decoding table. Its also evident that with this approach instead of '6' hops I will get at all the bits decoded with only '3' hops, each hop givinig me '2' bits. Similarly, for M=2, the hop count will further decrease to only 2 with three bits decoded at each hop. The step size can thus be varied to any value to even get all the bits on a single hop.

5. IMPLEMENTATION

In order to establish the authenticity of the proposed optimization in the Algorithm, I have done some simulations. Matlab is used as the simulation environment. Results are collected for three versions of the simulation, namely

- The uncoded version
- Viterbi Decoding without look-ahead
- Viterbi Decoding with Look-ahead

The parameters chosen for the simulation are listed in Table 2. For simplicity the results are shown for K=3, r=1/2, M=1. The logic behind is that if it's true for the simple case than depending on our requirement we can extend the parameters to suit our needs. Another important parameter is the soft-decision decoding, although it is also an optimization (over the hard-decision) but its effect are not discussed as it has nothing to do with the throughput. I am also using block processing of data instead of sample by sample processing. With block processing I have all the bits that need to be processed, so their branch metrics can be computed in the initial stage without waiting for them. This gives a lot of freedom for the look-ahead transformation.

I am not only recording the efficiency in time but also the performance of the proposed design. The reason being this may be possible that it may be running faster but running poor in the performance. In that case the speed will be useless, as it will be failing to fulfill its foremost purpose of Forward Error Correction. For this a BER comparison is done between the original version of VA and the optimized one.

```
for i = 1:length(recieved_input)/2

path_mat1 = path_mt_array(1,1) + br_mat(1,i);
path_mat2 = path_mt_array(2,1) + br_mat(4,i);
[best_mat(1,1) state]= min([path_mat1,path_mat2]);
surv_st(1,1)  = state;
end
```

Figure 5- Part of code w/o Look-Ahead

Figure 4 and Figure 5 shows a portion of the actual codes. One can also judge form these snippets that the loop is running for half the time in case of look-ahead. And in case of look-ahead a comparison is done between four possible options instead of only two.

```
for i = 1:length(recieved_input)/4
path_mat1 = path_mt_array(1,1) + br_mat(1,2*i-
1)+br_mat(1,2*i);

path_mat2 = path_mt_array(2,1) + br_mat(4,2*i-
1)+br_mat(1,2*i);

path_mat3 = path_mt_array(3,1) + br_mat(3,2*i-
1)+br_mat(4,2*i);

path_mat4 = path_mt_array(4,1) + br_mat(2,2*i-
1)+br_mat(4,2*i);

[best_mat(1,1) state] =
min([path_mat1,path_mat2,path_mat3,path_mat4]);
surv_st(1,1)  = state;
end
```

Figure 6- Part of code with Look-Ahead

Parameter	Value
Constraint Length	K=3
Data Length	10^5 samples
Generator polynomial	$(7,5)_8$
Rate	r=1/2
Look-Ahead step	M=1
Modulation	BPSK
	Block Processing
	Soft-decision
	Trace-back decdoing

Table 1- Simulation Parameters

It is also worth mentioning that both the versions of the VA are using block processing for a fair comparison.

6. RESULTS

Results are presented in this section. The results look very encouraging. The reported speed up Table 1, with only one look-ahead step (M=1) is roughly 50% reduction in execution time.

	Sequential VA	Optimized VA
Execution Time in seconds	38.3294	20.0305

Table 2- Simulation Results

The BER performance curve of the different simultaion sare shown in Figure. The curve clearly shows that the opimized design provides the speed up without any performance loss. The optimizaed BER performance is the same as that of the unoptimized one.

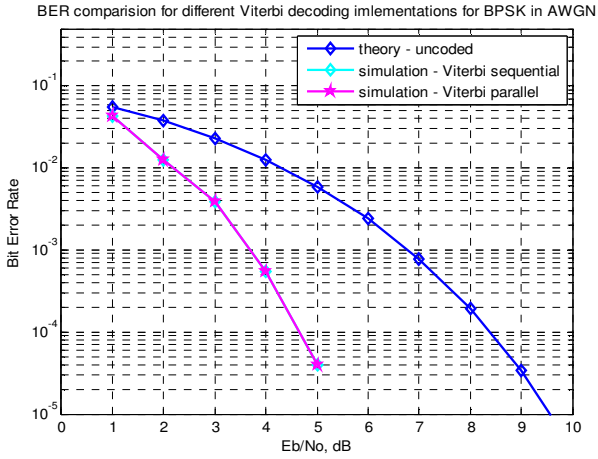

Figure 7- BER comparison of Actual and Proposed design

7. CONCLUSIONS

In this project look-ahead technique has been exploited to create the desired level of concurrency in the sequential design of Viterbi Algorithm. It's clear from the results that Viterbi decoding can be optimized using the Look-Ahead transformation technique to give a vectorized output. The choice of look-ahead step is application dependent. It's quite interesting that the decrease in execution time is extremely high with only a single look-ahead step. This may lead the designers to opt for this approach despite the expected large hardware requirements.

Although this algorithm can be implemented in a few lines of code, its storage space increases exponentially with each additional level of look-ahead. Also the computational complexity increases exponentially to compute the next set of matrices. For un-optimized VA the look-up table for decoding only require one bit of storage while in the case of look-ahead implementation the size of look up table increases exponentially with the increase of look-ahead step. Therefore, for large levels of look-ahead this algorithm would be hard to use. At the same time, the physical hardware needed to implement many levels of look-ahead would become a limiting time and space constraint.

8. FUTURE WORK

In this project I have considered a vectorized high throughput implementation of Viterbi decoder in Matlab, the next logical step would be to go for a hardware implementation of the proposed algorithm. This will be more challenging as while designing the optimized VA no consideration has been put on the hardware cost. The design considered in this project supports the binary convolutional codes only. It will be interesting to investigate the design for non binary case.

From the implementation point of view, the proposed design also faces the problem that it leads to high latency for look-ahead ACS precomputations. Parhi et al have looked into this matter but still more work needed to be done in this regard.

9. REFERENCES

[1] G.Fettweis and H. Meyr, "Parallel Viterbi decoding by breaking the compare-select feedback bottleneck" in Proc. IEEE Int. Conf. Communications, pp.719-723, June 1988.

[2] H. Thappar and J. Cioffi, "A block processing method for designing high-speed Viterbi decoders" in Proc. IEEE Int. Conf. Communications, pp. 836-840, 1989

[3] K.K. Parhi, "Look-Ahead in Dynamic Programming and Quantizer Loops" in IEEE Int. Conf. Circuits and Systems, 1989

[4] J.J.Kong and K.K.Parhi, "K-nested layered look-ahead method and architectures for high throughput Viterbi decoder," in Proc. 2003 IEEE Workshop on Signal Processing Systems, pp. 99-104

[5] K.K. Parhi, "High-Speed VLSI Architectures for Huffman and Viterbi Decoders", IEEE 1992

[6] M.F.Batcha and A.Z.Sha'ameri, "Configurable Adaptive Viterbi Decoder for GPRS, EDGE and Wimax" IEEE Conf., May2007K.K. Parhi, "High-Speed VLSI Architectures for Huffman and Viterbi Decoders", IEEE 1992

[7] S.Nandula,Y.S Rao, and S.P. Embanath, "High speed area efficient configurable Viterbi decoder for WiFi and WiMax systems, " June 2007

[8] G. Fettweis, L.Thiele, G.Meyr "Algorithm transformations for unlimited parallelism" IEEE Transactions on Very Large Scale Integration (VLSI) Systems (VLSI) SYSTEMS) 15 (2007).

[9] J.G. Back, S.H. Yoon, and J.W. Chong "Memory Efficient Pipelined Viterbi Decoder with Look-Ahead Traceback." IEEE (2001).

[10] G.D.Forney, Jr., "The Viterbi Algorithm," IEEE Proceedings, Vol.61, pp. 268-279, March 1973

[11] G.Fettweis and H. Meyr, "High-Speed Parallel Viterbi Decoding: Algorithm and VLSI-Architecture " IEEE Communication Magazine, May 1991

[12] H.D.Lin and D.G.Messerschmit, "Algorithms and architectures for concurrent Viterbi decoding," in Conf. Rec. 1989 IEEE Int. Conf. Communications, vol.2, pp. 836-840

[13] J.J.Kong and K.K.Parhi, "Low-Latency Architectures for High-Throughput Rate Viterbi Decoders," in Proc. 2004 IEEE Transactions on VLSI Systems, pp. 99-104

[14] D.G.Messerschmit and H.D.Lin" Arbitrarily High Speed Viterbi Decoders," UC Berkley, 1988

[15] C.Y.Chang and K.Yao. "Viterbi decoding by systolic arrays, " in Proc. Ann. Allerton Conf. Communications, Controls, and Computing, pp. 430-439, Oct 1985

[16] S.H. Choi and J.J.Kong, "State Parallel Viterbi Decoder Soft IP and Its Applications" IEEE 2001.

[17] S.C.Kim, J.H.Ryu, and J.D Cho, “Low Power , High Rate Viterbi decoder employing the SST Scheme and Radix-4 Trellis”