Final Project Proposal: The Viterbi Algorithm as a Stream Application

John Davis, Andrew Lin, Njuguna Njoroge, Ayodele Thomas
EE482C Advanced Computer Organization: Stream Processor Architectures
Stanford University, Spring 2002

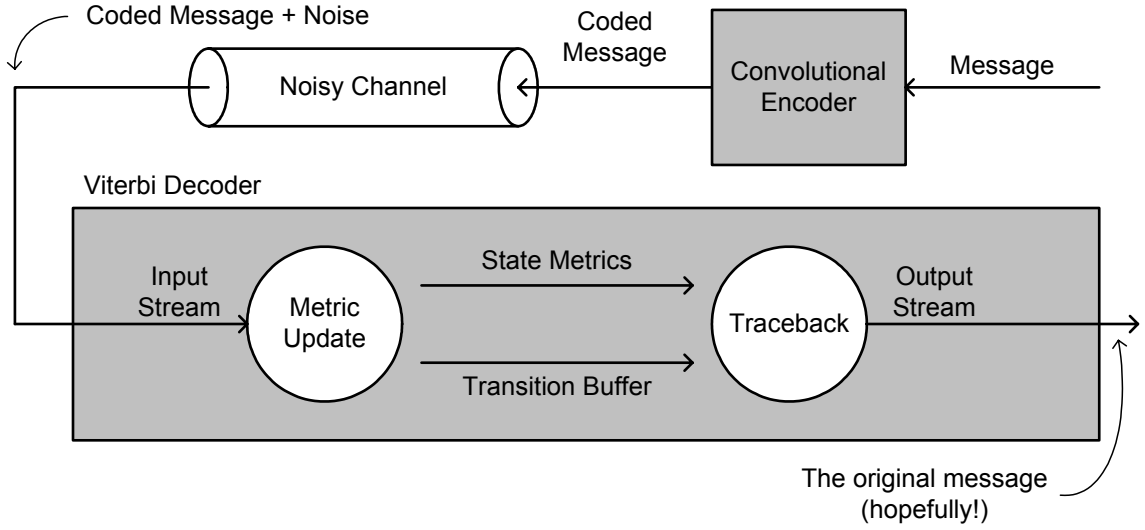
The Viterbi Algorithm

Background

Convolutional coding is a popular error-correcting coding method used in digital communications. A message is convoluted, and then transmitted into a noisy channel. This convolution operation encodes some redundant information into the transmitted signal, thereby improving the data capacity of the channel. The Viterbi algorithm is a popular method used to decode convolutionally coded messages. The algorithm tracks down the most likely state sequences the encoder went through in encoding the message, and uses this information to determine the original message. Instead of estimating a message based on each individual sample in the signal, the convolution encoding and Viterbi decoding process packages and encodes a message as a sequence, providing a level of correlation between each sample in the signal.

The Viterbi Algorithm as a Streaming Process

The Viterbi algorithm fits nicely into the streaming paradigm. For a ½ rate system, a stream of two-bit digital elements is input into the Viterbi decoder, and a stream of one-bit elements is returned. The decoding process involves two major steps – metric update and traceback. These two steps can be thought of as kernels. An illustration of how these two kernels might interconnect with input and output streams is shown below.



In this particular model, a stream that consists of the coded message and injected noise is input to the Viterbi decoder. Within the decoder, a Metric Update kernel is performed, which produces two streams – a state metric stream, which contains the accumulated state metrics for all delay states, and a transition stream, which contains the optimal path chosen for each delay state. These two streams are passed to a traceback stream, which traverses the state metric stream and employs the transition stream to find the optimal path through the Trellis. This illustrates one stream-based model of the Viterbi decode process.

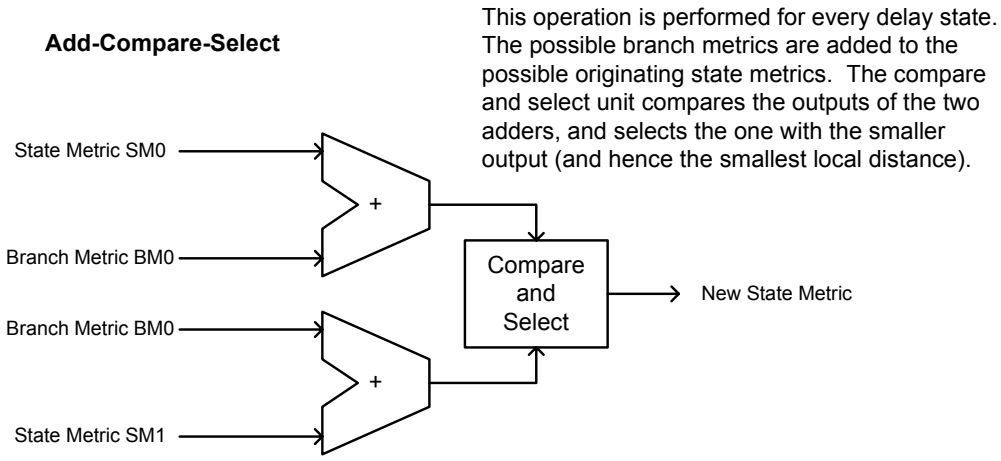
Implementing a Viterbi Decoder in Hardware

A Viterbi decoder can be implemented using a DSP [1] or as an ASIC [2]. Implementing the Viterbi decoder as an ASIC is more efficient in terms of power and performance. However, an ASIC is, for the

most part, a fixed design, and does not allow for much operational flexibility. A DSP provides a large amount of operational flexibility, since the Viterbi algorithm is implemented as a program, which is executed by the DSP. This flexibility is gained at the loss of performance and power efficiency.

We believe that the streaming processor architecture, with its many parallel clusters, and execution units within clusters, may provide higher power efficiency and performance than a typical DSP, yet more flexibility than an ASIC. For each delay state, an add-compare-select (ACS) operation is performed to determine the most probable path through the trellis. An ACS operation must be performed for all possible delay states. Parallelism between the ACS operations can potentially be exploited here. This parallelism can be mapped to the multiple clusters on Imagine. Another way to exploit parallelism is to run multiple decoders in parallel on blocks of data [2]. Perhaps a single cluster can be used as a decoder, therefore allowing multiple decoders to operate in parallel on a single Imagine chip.

We want to investigate the use of a streaming processor (Imagine) as a Viterbi decoder. We not only wish to compare the performance of Imagine with that of a DSP, but also identify the performance bottlenecks. The next sections describe the goals of our project in more detail.



Project Objectives:
The Viterbi algorithm appears to be a classic streaming application, and thus, it can be exploited by Imagine. Below, we have outlined the milestones and results that we hope to achieve from this project. We have located background information on the Viterbi algorithm, see the references below.

Milestones:

- Algorithm evaluation: Is this a streaming application?
- 16 state Viterbi encoder
- Assembly code for TI c54x DSP
- Partitioning methods: single viterbi across all clusters, single viterbi per cluster, etc.
- StreamC and KernelC implementation for Imagine
- Benchmark assembly code
- Benchmark Imagine code
- Analysis of Imagine schedule and performance results
- Conclusions/suggestions
- Try to generalize feedback implementation for stream processors

If time provides:

- Optimizing code
- Modifying .md file if appropriate
- Brook implementation

Results:

- Ease of programming, assembly vs. higher level language
- Performance comparison
- Exploited parallelism in Viterbi algorithm
- Imagine architecture critique

References

1. Hendrix, H., “Viterbi Decoding Techniques in the TMS320C54x Family,” Texas Instruments, June 1996.

2. P. Black, T. Meng, “A 140-Mb/s, 32-State, Radix-4 Viterbi Decoder,” *Journal of Solid State Circuits*, Vol. 27, No. 12, December 1992.

3. David Forney Jr., "The Viterbi Algorithm", Proceedings of the IEEE, Vol. 61, No. 3, March 1973