Project Parallel Viterbi Decoder

Yosi Ben Asher
Computer Science Dept.
University of Haifa, Haifa, Israel

1 Trellis generation via matrix multiplications.

- Figure 1 illustrates the trellis that needs to be built for a 1 $\longrightarrow$ 2-bit convolutional code of the presentation.
- Figure 2 illustrates the *vector* $\times$ *matrix* multiplication to compute the next stage of the trellis described by a two state automata DAG. The weights on the edges are the humming editing distance of the input and the state transitions.
- Figure 3 illustrates that the matrix multiplication works showing that by multiplying two transition matrixes we can correctly compute the accumulated hamming distances of a two-transitions trellis.

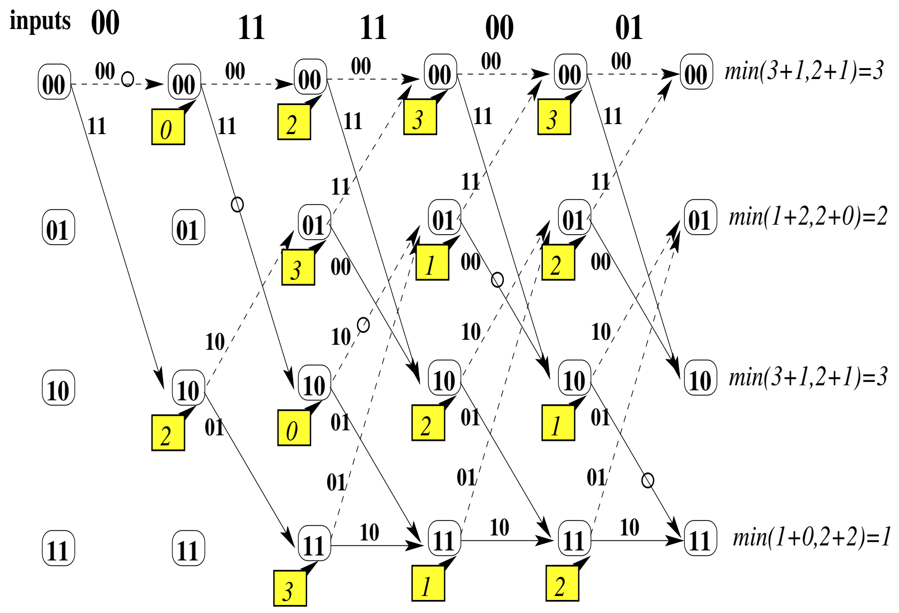
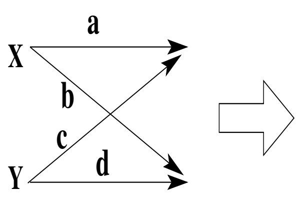
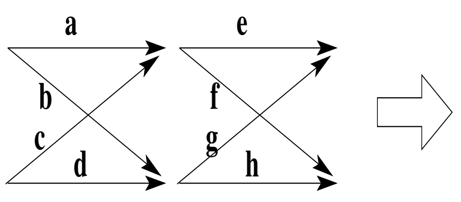


Figure 1: The trellis and accumulated hamming distance for a sequence of six inputs.

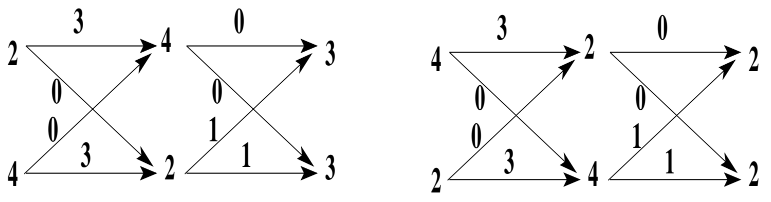


$$\begin{bmatrix} X \\ Y \end{bmatrix} \begin{pmatrix} a & c \\ b & d \end{pmatrix} = \begin{bmatrix} \min(X+a,Y+c) \\ \min(X+b,Y+d) \end{bmatrix}$$



$$\begin{pmatrix} a & c \\ b & d \end{pmatrix} * \begin{pmatrix} e & g \\ f & h \end{pmatrix} = \begin{pmatrix} \min(a+e,b+g) & \min(c+e,d+g) \\ \min(a+f,b+h) & \min(c+f,d+h) \end{pmatrix}$$

Figure 2: Representing the trellis as a matrix and the concatenation of two trellis as a matrix multiplication.



$$\begin{pmatrix} 3 & 0 \\ 0 & 3 \end{pmatrix} * \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \min(3+0,0+1) & \min(0+0,3+1) \\ \min(3+0,0+1) & \min(0+0,3+1) \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}$$

$$\begin{bmatrix} 2 \\ 4 \end{bmatrix} \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} = \begin{bmatrix} 3 \\ 3 \end{bmatrix} \qquad \begin{bmatrix} 4 \\ 2 \end{bmatrix} \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

Figure 3: Two examples demonstrating the matrix multiplication approach.

2 C-part

Write a program in C that the following components:

1. Write a C/C++ program that has the following constants: n-total number of input bits, k, l- k input bits are stretched to $l > k$ output bits. m - number of registers, each register is of k bits and there is a shift operation over these registers every new k-bit inputs is read. p - degree of parallelism.

2. Write the encoder as a general function that gets the input bits in an array and output encoding into an output array of $(n/k) * l$ bits. The output is derived by l XORs whose inputs are selected randomly from the $m * k$ bits that are stored in the m registers.

3. Copy the encoded sequence to a different array and add some errors to the codded sequence.

4. Write a function that, given the encoder generates the equivalent automata A, with w states.

5. Write the sequential-decoder that uses A to build an $n \times w$ trellis. The encoder should process the the trellis computing the accumulated hamming distance and selecting the minimum-path and correcting the codded sequence (as is described in the slides).

6. Test that your code works o.k. and can correct bits.

7. Write the parallel decoder that uses parallel matrix multiplication technique to compute the trellis in parallel. Use an external $for(i = 0; i < n; i+ = n/p)$...... to represent p threads running in parallel. If it helps you can set the constants such that n/p will be a power of two. Note that though parallel matrix multiplication allow computing the accumulated hamming distance in parallel you still have to figure how to compute the minimal path in parallel.

8. When both the sequential and parallel decoder works well, then transform your program such that it will print the sequential and parallel decoders as Verilog module including the encoded sequence. You can pass the matrixes directly rom teh C program to the Verilog modules, however at the next stage compute the transiton hamming distance (edge weights) directly from the array of inputs.

9. Evaluate the resulting program showing that speedups can be obtained for sufficiently large n. Determine the power and timing of the sequential and parallel decoders and compute an optimized configuration of your parallel decoder that minimizes power and time.