Viterbi Algorithm on a Hypercube: Concurrent Formulation

F. Pollara
Communications Systems Research Section

The similarity between the Fast Fourier Transform and the Viterbi algorithm is exploited to develop a Concurrent Viterbi Algorithm suitable for a multiprocessor system interconnected as a hypercube The proposed algorithm can efficiently decode large constraint length convolutional codes, using different degrees of parallelism, and is attractive for VLSI implementation.

I. Introduction

Concurrent computers have the potential to obtain large increases in computational power. This is true if one can find a concurrent decomposition of a given algorithm.

We consider the Viterbi Algorithm for decoding $(m, 1/n_0)$ convolutional codes, where m is the memory (constraint length = $m + 1$) and $1/n_0$ is the code rate, and we describe an efficient decomposition of this problem, the Concurrent Viterbi Algorithm (CVA), which is suitable for multiprocessor systems with a Hypercube architecture.

There are two key requirements in the problem decomposition:

(1) Divide the problem in equal parts, in order to share equally the execution time available in each processor.

(2) Minimize the communication between the parts, so that each processor needs to share information only with its neighbors.

Fox (Ref. 1) has shown that the Hypercube is a natural topology for the binary Fast Fourier Transform (FFT), and high efficiency can be obtained with this structure. We will show that there is a simple correspondence between the connectivity of the FFT algorithm and the trellis diagram of the Viterbi Algorithm. Therefore, efficient methods for implementing the Viterbi Algorithm on a Hypercube computer can be developed.

II. Hypercube Computer Structure

In general, it is desirable for an interconnection structure to have a low number of links per node (low degree of a node), and a small internode distance. This distance d_{xy} between any two nodes x and y is defined as the minimum number of links required to send a message from x to y. The diameter D of a network with N nodes is defined as $D = \max\{d_{xy} | 0 \leq x, y \leq N\}$.

A Boolean n-cube computer, or just Hypercube (Ref. 2), is a network of $N = 2^n$ processors placed at the vertices of an n-cube, and connected by its edges. Such a structure is attractive because both the degree of a node and the diameter, which are equal to n, grow only linearly with the dimensionality of the cube.

In a multiprocessor network there is always a trade-off between the degree of a node and the diameter, the extreme

case being the completely connected network which has $N-1$ links per node and $D=1$. In general a low degree of a node implies a large diameter.

The Hypercube structure seems a reasonable compromise for practical multiprocessor systems, except for the disadvantage of offering limited input/output capability This negative aspect, which is less important for computationally intensive (vs. I/O intensive) algorithms, is furthermore mitigated by the small diameter of the network, and the simplicity of efficient broadcasting methods.

Two processors x and y will be called neighbors if their binary labels differ only in one position, i.e.,

$$x = [x_{n-1}, x_{n-2}, \ldots, x_k, \ldots, x_0]$$

$$= \sum_{k=0}^{n-1} 2^k x_k$$

where $x_k \in \{0,1\}$, $y = [x_{n-1}, x_{n-2}, \ldots, \bar{x}_k, \ldots, x_0]$, and $\bar{x}_k$ is the complement of x_k. In this context, the distance d_{xy} between two processors is just the Hamming distance of their binary labels, and neighbors have $d_{xy}=1$. In the Hypercube, the number of nodes at distance d from a given node (node $[0,0,\ldots,0]$ can be considered without loss of generality) is $N_d = \binom{n}{d}$, so that the average distance is

$$\bar{d} = \frac{1}{N-1} \sum_{d=1}^{n} d \binom{n}{d}$$

$$= \frac{n}{2} \frac{N}{N-1} \approx \frac{n}{2}$$

III. Equivalence of Networks

We now define two basic network structures the m-stage Viterbi algorithm trellis (de Bruijn graph), shown in Fig. 1 for $M = 2^m = 8$ states, and the FFT decimation-in-time graph of Fig. 2.

Let $x = [x_{m-1}, \ldots, x_0]$ be the binary index of a state or node of the graphs, and $u = [u_0, \ldots, u_k, \ldots, u_{m-1}]$, $u_k \in \{0,1\}$, be a sequence of input bits which define one of the two possible paths out of each node, where k represents the stage.

First, consider the elementary transformation $\sigma(x, u_k)$ which describes the state transitions at stage k, and is defined as

$$\sigma(x, u_k) = \sigma([x_{m-1}, \ldots, x_0], u_k)$$

$$\triangleq [u_k, x_{m-1}, \ldots, x_1] \quad (1)$$

This is a cyclic right shift of x, with x_0 replaced by u_k. Then the transformation $\sigma(\cdot, \cdot)$ completely describes the transitions of the graph in Fig. 1, and the m-stage transformation from a state x at stage 0 to state y at stage m is defined as

$$y = \sigma^{(m)}(x, u)$$

$$\triangleq \sigma(\sigma(\ldots\sigma(\sigma(x, u_0), u_1)\ldots), u_{m-1}) \quad (2)$$

The elementary transformation needed to describe Fig 2 is given by

$$\omega(x, u_k) \triangleq [x_{m-1}, \ldots, x_{k+1}, u_k, x_{k-1}, \ldots, x_0] \quad (3)$$

which replaces x_k by u_k. The complete m-stage graph of Fig. 2 is then described by the transformation $\omega^{(m)}(x, u)$,

$$y = \omega^{(m)}(x, u)$$

$$\triangleq \omega(\omega(\ldots\omega(\omega(x, u_0), u_1)\ldots), u_{m-1})$$

$$= [u_{m-1}, \ldots, u_0] \quad (4)$$

Now, it is easy to verify that, at the mth stage,

$$\sigma^{(m)}(x, u) = \omega^{(m)}(x, u)$$

Therefore, the two networks lead from a given state x to the same state y, after m stages.

The equivalence of the two networks can be further expressed, at any stage, by defining the cyclic right shift operator

$$\rho^{(k)}(x) = [x_{k-1}, x_k, \ldots, x_{m-1}, x_0, \ldots, x_{k-2}]$$

where $\rho^{(m)}(x) = \rho^{(0)}(x) = x$, and verifying that,

$$\sigma^{(k)}(x, u) = \rho^{(k)}(\omega^{(k)}(x, u)) \quad (5)$$

This result shows that, if we relabel each node x of the graph in Fig. 1 with the label $\hat{x} = \rho^{(k)}(x)$ at stage k, the two networks are functionally and topologically equivalent; that is, they are just two different ways of drawing the same network. A given path generated by an input sequence u visits the same nodes in Fig. 1 and Fig. 2, if we relabel each node x in Fig. 1 by $\rho^{(k)}(x)$.

Having established this formalism on networks, we can now apply the above results to the study of algorithm structures, in particular to the Viterbi algorithm and its relationship with the radix-2 FFT.

IV. Concurrent Viterbi Algorithm

Consider a multiprocessor system with N processors located at the vertices of an n-cube and linked only by the edges of the cube (see an example for $n = 3$ in Fig. 3[a]). Note that the nodes are labeled by an n-bit binary number, so that the ith bit is the coordinate of a node along the ith dimension.

If in Fig. 2 we collapse the horizontal dimension, we obtain a graph which is exactly identical to that of Fig. 3, i.e., with the same connections between nodes. This observation, as explained in Refs. 1 and 3, suggests a natural way to implement the FFT on a Hypercube computer.

The implementation of the FFT on the Hypercube can be stated more formally if we define the Hypercube (m-cube) network by the transformation,

$$\gamma[x_{m-1}, \ldots, x_k, \ldots, x_0] = [x_{m-1}, \ldots, \bar{x}_k, \ldots, x_0] \tag{6}$$

where $\bar{x}_k$ is the complement of x_k, and observe that the transformation in Eq. (3) can always be obtained by Eq. (6), since $u_k \in \{0, 1\}$ and $u_k \equiv x_k$ requires no communication (self-loop).

To perform the first stage of Fig. 2, let the nodes of Fig 3 communicate along the first dimension, and so on for each stage and dimension. In this way, the links provided by the n-cube are just those necessary to perform the FFT. This implementation on the n-cube is possible since the FFT requires communication only between neighboring nodes of the cube.

At first, the network of Fig. 1 might seem to require communication between distant nodes on the cube. But this problem can be easily overcome if we relabel the nodes as discussed in Sec. III. Specifically, at stage k, processor x will represent state $\rho^{(k)}(x)$ of the Viterbi trellis. Processor transitions are described by the graph of Fig. 2, while state transitions are given by Fig. 1, as desired. Therefore a Viterbi decoder can be efficiently implemented on the Hypercube, exactly as for the FFT. The similarity between the FFT and Viterbi trellis was previously pointed out by Forney (Ref. 3), but apparently not exploited in any practical way.

Each processor, at stage k, receives the accumulated metric and survivor of its neighbor along dimension k of the cube, and performs the usual comparison and update. In a practical Viterbi decoder for a $(m, 1/n_0)$ code, the number of stages required to obtain a performance very close to optimum is approximately $5m$. Notice that, when the stage k is a multiple of m, the state and processor labels are identical, since $\rho^{(m)}(x) = x$, so that we may easily select the decoded bit belonging to the most likely survivor. However, in order to minimize internode communications, it may be more advantagenous to increase slightly the number of stages and read the decoded bit at node zero, which simplifies I/O operations (see Sec. V).

V. Message Broadcasting

Performing the CVA requires that blocks of data be loaded in every processor: This operation is called broadcasting. In the Hypercube (Ref. 2), data from the host processor is directly exchanged only through node zero (the origin of the cube). Therefore, an efficient concurrent method is required to broadcast a message from node zero to all other nodes. Since the diameter of an n-cube is n, a lower bound on the broadcasting time is n time units (where the unit is the time to send a message to a neighbor).

Assume that message A is in node zero, at time zero In each subsequent time slot k, send messages in parallel from each node $x = [x_{m-1}, \ldots, x_{k+1}, 0, x_{k-1}, \ldots, x_0]$ to each node $\gamma(x)$, the neighbors along dimension k. After n time units, message A has propagated to all nodes.

Even though this method does not minimize the number of communications (with the advantage of a very simple indexing), it optimizes the total broadcasting time to n time units. The result is clearly optimum, since it achieves the lower bound.

VI. Decoders for Large Constraint Length

Existing Hypercube computers have up to 128 nodes ($n = 7$), and will soon be extended to 1,024 nodes ($n = 10$). Yet, in order to decode powerful convolutional codes with $m > 10$, one needs to obtain algorithms which assign more than one state per processor. This need is dictated not only by the practical limitations on the physical size of the computer, but also by the goal of achieving high computational efficiency.

The efficiency η of a parallel computer is defined as,

$$\eta = \frac{N_o t_o}{N_o t_o + N_t t_t} = \frac{\text{sequential algorithm time}}{N \times (\text{parallel algorithm time})}$$

and the speed-up σ is given by $\sigma = \eta N$, where N is the number of processors, N_o is the number of parallel operations (butterflies), t_o is the time required by each operation, N_t is the number of parallel data transfers, and t_t is the communication time.

When the number of states M of the decoder is larger than the available number of processors N, states can be grouped in sets of $S = 2^s$ states per processor, where $M = SN$. To see how this is possible for $S = 2$ in the proposed CVA implementation, consider Fig. 2 and group each pair of nodes into one processor $P_{\lfloor i/2 \rfloor}$, for all nodes i, $i = 0, \ldots, 7$, where $\lfloor j \rfloor$ is the largest integer less than or equal to j. Similarly, with two processors $P_{\lfloor i/4 \rfloor}$, $i = 0, \ldots, 7$, we obtain the case $S = 4$. The extreme cases, $S = M$ and $S = 1$ represent the completely sequential and completely parallel decoder, respectively. Intermediate cases represent different degrees of parallelism or a different *granularity* of the algorithm, which is defined as the amount of computation between successive data transfers. In general, the efficiency increases with the granularity.

During m stages of the CVA, the number of parallel data transfers is,

$$N_t = S(m - s) = \frac{M}{N} n \tag{7}$$

since only $m - s$ stages of Fig. 2 need to communicate with neighbors (s stages do only internal computations). The number of parallel operations (butterflies) N_o is given by

$$N_o = \frac{S}{2} m = \frac{M}{2N} m \tag{8}$$

As an example, N_t and N_o are plotted in Fig. 4 for $M = 64$. From Eqs. (7) and (8) we have that the efficiency η decreases as S decreases from N to 1, and, for $N = M$, η is equal to the ratio $t_o/(t_o + 2t_t)$, which depends on the hardware implementation.

The CVA with $S > 1$ is useful because it allows the implementation of complex decoders, avoiding the pin-limitation constraint problem encountered in existing VLSI decoders. This problem is due to the particular partitioning of the traditional decoder into a branch metric computation block and a path memory storage block. This partition requires a rapidly increasing number of pins in the VLSI chips. The proposed CVA has instead a number of connections per node increasing only linearly with the dimensionality of the cube, and is therefore more suitable for VLSI implementation

Furthermore, the CVA has been extended to high rate $(m, k_0/n_0)$ codes with $k_0 > 1$, where k_0 is the number of input bits in the encoder. This extension is possible only for a limited range of m, k_0 and S, the number of states per processor. An example for $(3, 2/n_0)$ codes is given in Fig. 5, where $S = 2$.

Given a rate $1/n_0$ code it is known how to generate all the punctured codes of rate k_0/n_0, $k_0 > 1$. Since these codes involve only pairwise comparisons at each stage, it is certainly possible to decode them with the CVA. It must be noted however that punctured codes require more stages, and this is equivalent to linking nodes of the Hypercube which are not neighbors, by using multiple stages.

VII. Conclusion

The proposed CVA decoder has been implemented and successfully tested on a 64-node Hypercube computer for $(m, 1/n_0)$ codes, with $m = 2, \ldots, 14$. Present results confirm the usefulness of the CVA for large constraint length codes.

Acknowledgment

The author wishes to thank R. J. McEliece and C. L. Seitz of Caltech for providing access to the Hypercube and E. Majani for some of the programs.

References

1. Fox, G. C., and Otto, S. W., "Algorithms for Concurrent Processors," *Physics Today*, Vol. 37, No. 5, pp. 50-59, May 1984.
2. Seitz, C. L., "The Cosmic Cube," *Communications of the ACM*, Vol. 28, No. 1, pp. 23-33, January 1985
3. Pease, M. C., "An Adaptation of the Fast Fourier Transform for Parallel Processing," *Journal of the ACM*, Vol. 15, No. 2, pp. 252-264, April 1968.
4. Forney, G. D., "The Viterbi Algorithm," *Proc. IEEE*, Vol 61, No 3, pp. 268-278, March 1973.

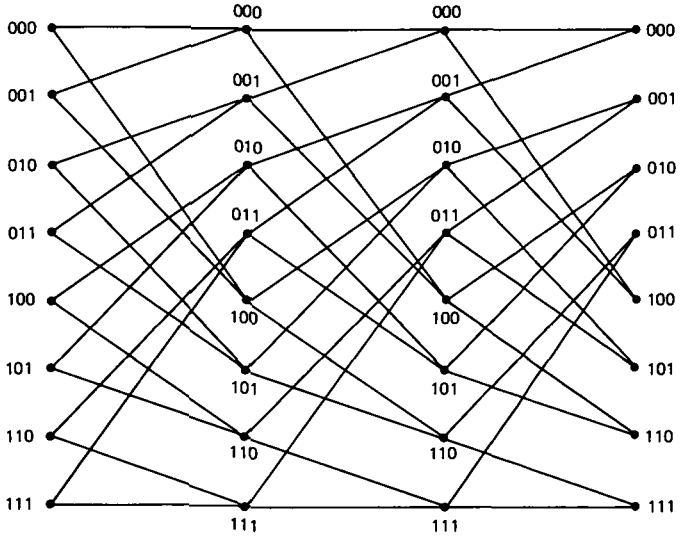

Fig. 1. The *m*-stage Viterbi algorithm trellis (*m* = 3)

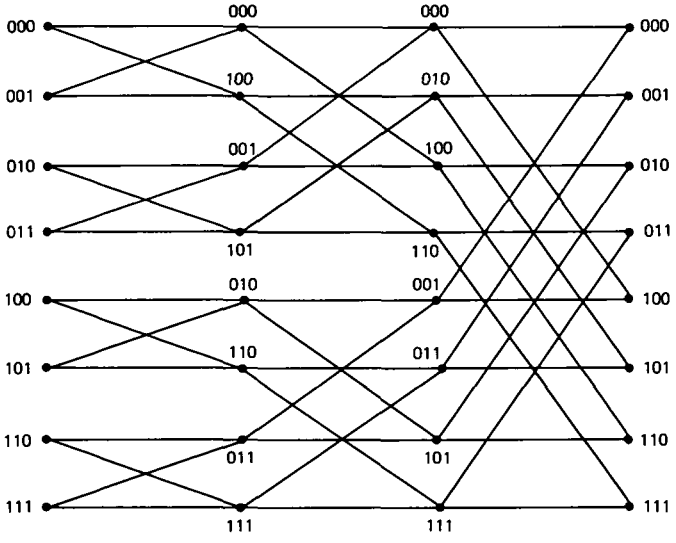

Fig. 2. The FFT decimation-in-time graph (*m* = 3)

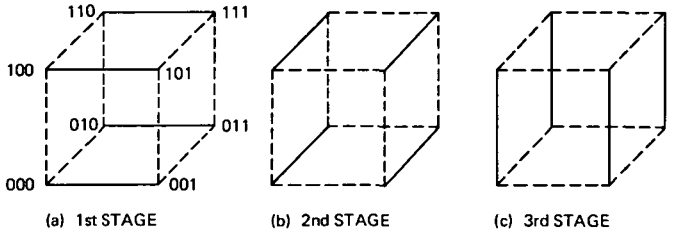

Fig. 3. The 3-cube graph

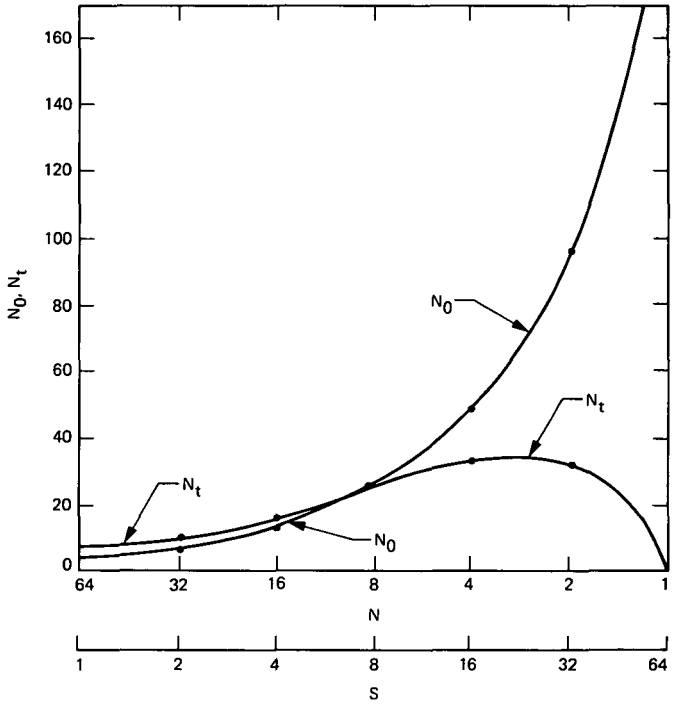

Fig. 4. N_0 and N_t vs S and N

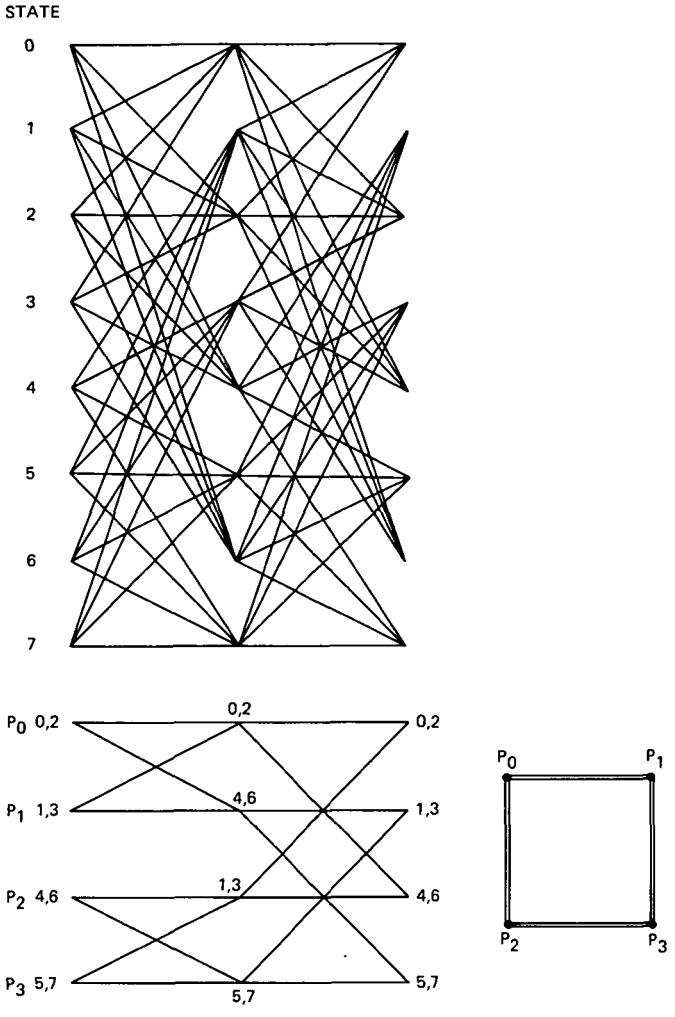

Fig. 5. Decoder for $(3, 2/n_0)$ code, with $S = 2$