Area-Efficient Architectures for the Viterbi Algorithm—Part I: Theory

C. Bernard Shung, Horng-Dar Lin, Robert Cypher, Paul H. Siegel, and Hemant K. Thapar

***Abstract*—The Viterbi algorithm has been widely applied to many decoding and estimation applications in communications and signal processing. A state-parallel implementation is usually used in which one add-compare-select (ACS) unit is devoted to each state in the trellis. In this paper we present a systematic approach of partitioning, scheduling, and mapping the N trellis states to P ACS's, where $N > P$. The area saving of our architecture comes from the reduced number of both the ACS's and interconnection wires. The design of the ACS, path metric storage, and routing network is discussed in detail. The proposed architecture creates *internal parallelism* due to the ACS sharing, which can be exploited to increase the throughput rate by pipelining. Consequently, the area-efficient architecture offers a favorable (smaller) area-time product, compared to a state-parallel implementation. These results will be demonstrated by application examples in the accompanying paper.**

I. Introduction

The Viterbi algorithm [8], [13] has been widely applied to many decoding and estimation applications in communications and signal processing. It can be used to perform maximum likelihood decoding for convolutional codes, or maximum likelihood estimation of the transmitted sequence over a channel with intersymbol interference (ISI) [7]. Recently, it was also applied to hidden Markov model based continuous speech recognition [11].

The Viterbi algorithm is defined in terms of a *trellis diagram.* In decoding, the algorithm attempts to reconstruct the actions of the encoder based on the transmission of its outputs over a noisy channel. Specifically, it assigns a value, called the *path metric,* to each node in the trellis. For each incoming branch, the path metric of the node from which the branch originated is added to a *branch metric* which gives the likelihood that the corresponding state transition occurred, based on the value received over the noisy channel. The sums calculated for each of the incoming branches are compared and the largest one (corresponding to the most likely transition) is selected.[1] This selected sum is used as the new path metric for the node. The calculation of the path metrics is often implemented by an Add-Compare-Select (ACS) unit.

Because of the feedback loop, the ACS is generally considered to be the most critical part in the implementation of the Viterbi algorithm. Most implementations employ a *state-parallel* approach in which one ACS is devoted to each state in the trellis. The speed bottleneck in such implementations is the delay of the ACS. In continuous speech recognition in which the number of states is large and the sampling rate is low, a *state-serial* approach is often used. All the states share one (or a few) ACS sequentially. In this paper, we study the general problem of partitioning and scheduling the states in sharing the ACS's. The proposed *area-efficient* architecture is shown to provide favorable area-time tradeoffs in practical examples.

Several architectures have been proposed to address the speed/area tradeoff of the ACS. Fettweis and Meyr [3] (and, independently, Lin and Messerschmitt [10] and Thapar and Cioffi [12]) proposed a technique to trade area for speed, in which speed beyond the ACS bottleneck was achieved by using a parallel implementation that operates on the *M-Step* trellis. Gulak and Shwedyk [6] observed that the state-parallel implementation with a de Bruijn graph trellis results in an interconnection network defined by a shuffle-exchange graph. Based on the VLSI grid model, they applied the work on optimum layout of shuffle-exchange graphs to reduce the wiring area. Gulak and Kailath [5] proposed three types of locally connected processor arrays—*cascade, linear,* and *mesh*—for trellises with a de Bruijn graph structure. The idea was to further reduce wiring area by limiting each ACS to communicate only with its neighbors. In linear and mesh arrays, N ACS's are required for a trellis of N states. In a cascade array, *logN butterfly* processors are required for a trellis of N states. However, each of the *logN* processors serves all N states in sequence with complex switches used among the processors for timing alignment.

In this paper, we propose a new area-efficient architecture model that trades speed for area. One unique feature of our scheme is that each ACS is shared by a fixed subset of the states. The novelty of our architecture is best illustrated by the area-time diagram (AT diagram) in Fig. 1. The state-parallel implementation of a particular trellis corresponds to a fixed point in the AT diagram. It can be seen that the high-speed techniques [3], [10], [12] and the area-efficient techniques can be used to explore the design space in two *different* directions.

Paper approved by the Editor for VLSI in Communications of the IEEE Communications Society. Manuscript received August 29, 1990; revised April 14, 1991. This paper was presented in part at the Global Telecommunications Conference, San Diego, CA, December 1990.

C. B. Shung is with the Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, Republic of China.

H. D. Lin is with AT&T Bell Laboratories, Holmdel, NJ, 07733.

R. E. Cypher and P. H. Siegel are with the IBM Research Division, Almaden Research Center, San Jose, CA, 95120-6099.

H. K. Thaper is with the IBM Storage Systems Products Division, San Jose, CA, xxxxx.

IEEE Log Number 9209473.

[1] Alternately, The *distance* or *inverse likelihood* can used as metric. In this case, the most likely transition corresponds to the smallest sum.

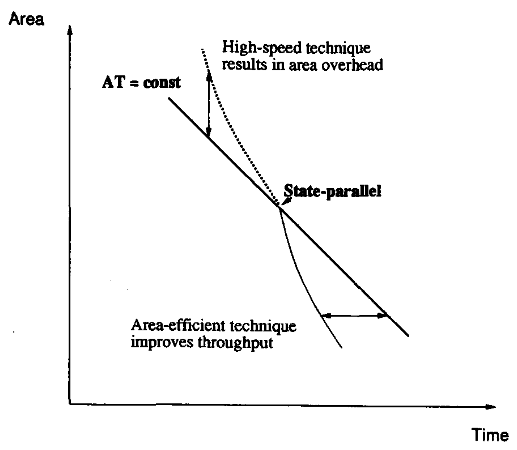


Fig. 1. Area-time (AT) diagram. **AT = const** is used as the reference line. The area-efficient technique (solid curve) can be used to trade speed for area saving, while the high-speed technique (dashed curve) can be used to trade area for speed. Comparing with the reference, it can be seen that the area-efficient technique improves throughput, while the high-speed technique results in area overhead.

We choose $AT = constant$ to be the reference curve.[2] The complexity of the M-*step* ACS in [3] introduces *area overhead* in the high-speed techniques. On the other hand, our area-efficient techniques provide *throughput increase,* by virtue of the introduction of internal parallelism. Furthermore, the area-efficient techniques based on locally connected processor arrays [5] suggest *discrete* design choices in the AT diagram. On the contrary, our technique provides a *continuous* means of trading speed for area. It is therefore easier to find an optimal design solution for particular area or speed constraints.

The paper is organized as follows. In Section II, we introduce our area-efficient architecture model. We define several criteria that we use in the paper to evaluate the architecture. In Section III, trellis partitioning and scheduling are described. In Section IV, we discuss the design of the local memory where the path metrics are stored. Design techniques to achieve higher throughput rate with a pipelined ACS are discussed in Section V. The area-time analysis is provided in Section VI. Applications examples of the area-efficient architectures will be shown in the accompanying paper.

II. Architectural Model for Area-Efficient Implementation

Fig. 2 illustrates our architectural model for area-efficient implementation of the Viterbi algorithm. It consists of P ACS's with associated local memories and a routing network. We assume that the N trellis states are evenly partitioned into the P ACS's, where P divides N.[3] We call the implementation *time-efficient* if no ACS is idle at any time, so it takes (N/P) time units to finish all N trellis states. We define the *i-degree,* d_i, of a trellis state as the number of incoming edges

[2]In Fig. 1, both area and time axes are in a logarithmic scale, therefore, $AT = constant$ is represented by a straight line.

[3]If P does not divide N, then $P\lceil N/P\rceil - N$ ACS's will be idle in one of $\lceil N/P\rceil$ time units, assuming one time unit is required for one ACS to process one state.

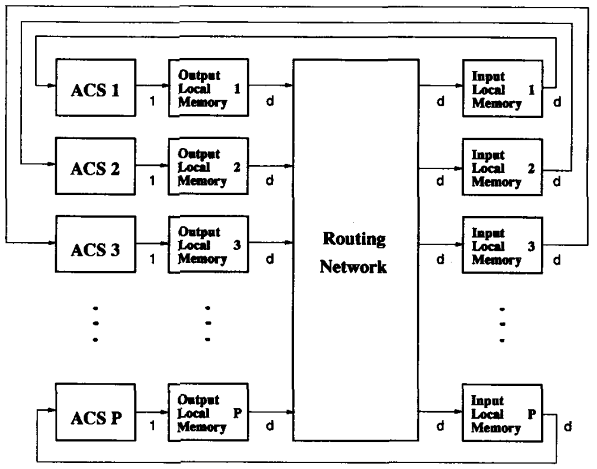


Fig. 2. The architecture model for the area-efficient implementation.

to that state. We define the *o-degree,* d_o, of a trellis state as the number of outgoing edges from that state. We call a trellis *homogeneous* if for all states, $d_i = d_o = d$, and we call d the *degree* of the trellis. We will concentrate on homogeneous trellises, as nonhomogeneous trellises can be made homogeneous by adding dummy branches.

The local memory of each ACS is divided into two banks—one storing the (dN/P) old path metrics (the inputs to the ACS), and the other storing the (N/P) new path metrics (the outputs of the ACS). The two local memory banks are referred to as the *input local memory* and the *output local memory.* The input local memory has d read ports and d write ports, and the output local memory has d read ports and 1 write port. In Section V, we will discuss alternative design techniques for the local memories to avoid the need for multiport memories.

For a homogeneous trellis of N states and degree d, the P ACS's together require Pd old path metrics at any one time. We are interested in finding an implementation in which the Pd old path metrics which enter the routing network at a given time are evenly distributed among the P local memories. In such a case, only d wires are required from each local memory to the routing network. We call such a design *bandwidth-efficient* as the bandwidth of the routing network is reduced from Nd (in a state-parallel implementation) to Pd.

In a bandwidth-efficient implementation, the routing network routes Pd old path metrics to the input local memory of the P ACS's at each time unit. In general, the required routing (*permutation*) is different in each time unit, and hence a *dynamic* routing network has to be used. We will discuss the use of a *multistage interconnection network* (MIN) [14] in this case. If the required routing is the same in each time unit, the routing network degenerates to Pd wires. We call this the *fixed-interconnection* case. In Section IV, we discuss partitioning and scheduling techniques for both fixed-interconnection and MIN routing networks.

Based on the architectural model, the goal is to find a time-efficient and bandwidth-efficient implementation for a particular trellis diagram. It will be shown in Section IV that bandwidth-efficiency can always be achieved by our partitioning, scheduling, and architecture mapping techniques. It will be shown in Section VI that time efficiency can always be achieved if interleaved independent data sources are allowed. We will also show that the sharing of an ACS allows additional pipelining of the states which can be exploited to improve the throughput rate. This, plus the area reduction enabled by reducing the number of ACS's and interconnections, produces a favorable area-time tradeoff for the proposed architecture.

III. Trellis Partitioning and Scheduling

In this section we discuss how to partition the N trellis states into P ACS's and schedule the (N/P) states within each ACS, where P is chosen depending upon the amount of area saving desired. We will begin by assuming the existence of a MIN routing network, and we will show that *any* trellis can be implemented in a bandwidth-efficient manner. We will then give the criterion which a trellis must meet in order for the MIN routing network to be replaced by a fixed-interconnection network.

A. Partitioning, Scheduling, and Communication Requirements

The partitioning of the N trellis states into the P ACS's, and the scheduling of the (N/P) states within each ACS, have a major impact on the design of the routing network and the ACS local memories. The matrix $S = \{s_{ij}\}$ will represent the partitioning and scheduling of the trellis states. Specifically, let s_{ij} be the state that is assigned to the ith ACS and scheduled to be processed at the jth time unit, $0 \leq i < P$, $0 \leq j < (N/P)$.

Given the partitioning and scheduling matrix S and the trellis which is being used, we can define two additional matrices, X and Y, which capture the communication requirements. Matrix X is the Pd-by-(N/P) matrix created by making d copies of each row of S. Specifically, $x_{ij} = s_{kj}$, where $k = \lfloor i/d \rfloor$. Thus, if d copies are made of each ACS output, x_{ij} represents the ith output from the array of ACS's at the jth time unit. Matrix Y is the Pd-by-(N/P) matrix created by replacing each entry $s_{ij} \in S$ with a column of d entries giving the d old path metrics which are needed in order to calculate s_{ij}. Specifically, $\{y_{ij} \mid \lfloor i/d \rfloor = k\}$ is the set of d old path metrics that are needed to calculate the new path metric for state s_{kj}. Thus, y_{ij} represents the ith input that is required by the array of ACS's at the jth time unit.

Example 1: A 6-state degree-2 homogeneous trellis is shown in Fig. 3. The S, X, and Y matrices of a particular partitioning and scheduling are shown below.

$$S = \begin{pmatrix} 0 & 4 & 2 \\ 3 & 1 & 5 \end{pmatrix} \qquad X = \begin{pmatrix} 0 & 4 & 2 \\ 0 & 4 & 2 \\ 3 & 1 & 5 \\ 3 & 1 & 5 \end{pmatrix}$$

$$Y = \begin{pmatrix} 0 & 4 & 2 \\ 5 & 3 & 1 \\ 2 & 0 & 4 \\ 3 & 1 & 5 \end{pmatrix}.$$

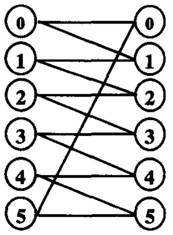


Fig. 3. A 6-state degree-2 trellis for Example 1.

Because the outputs of the array of ACS's are created in the pattern defined by X, and the inputs to the array of ACS's are required in the pattern defined by Y, the architecture must support the mapping of the matrix X into the matrix Y. A permutation which accomplishes this mapping from X to Y will be called a *matrix permutation* for the given trellis. This matrix permutation will be implemented in the routing network and/or the local memories of the ACS's.

The next subsection presents a general technique for dividing a matrix permutation into a number of smaller, independent permutations. This general technique will then be used to obtain a bandwidth-efficient implementation of an *arbitrary* matrix permutation on our architectural model. Therefore, an arbitrary partitioning and scheduling matrix S can be chosen. However, we will show later that the choice of S in a fixed-interconnection implementation can be difficult.

B. Matrix Permutations

In this subsection we will show that any matrix permutation can be divided into three successive permutations—the first and third of which only rearrange elements within rows, and the second of which only rearranges elements within columns. We will need the following definitions.

Definitions: $\mathcal{M}$ is the set of Pd-by-(N/P) matrix permutations. Thus, each member of $\mathcal{M}$ is a function which takes as input a Pd-by-(N/P) matrix, and creates as output another Pd-by-(N/P) matrix which is obtained by rearranging the elements of the input matrix. $\mathcal{R}$ is the set of matrix within-rows-permutations, and $\mathcal{C}$ is the set of matrix within-columns-permutations. Thus, each member of $\mathcal{R}(\mathcal{C})$ is a function which takes as input a Pd-by-(N/P) matrix, and creates as output another Pd-by-(N/P) matrix which is obtained by rearranging the elements in each row (column) of the input matrix. The permutation obtained by applying permutation A_1 and then applying permutation A_2 will be denoted $A_2 \circ A_1$.

The following theorem is due to Benes [1].

Theorem 1: For each permutation $M \in \mathcal{M}$, there exist permutations A_1, A_2, and A_3 such that $A_1 \in \mathcal{R}, A_2 \in \mathcal{C}, A_3 \in \mathcal{R}$, and $M \equiv A_3 \circ A_2 \circ A_1$.

Example 2: Here is a simple 4-by-4 example of Theorem 1. The permutation can be defined by a matrix of numbers from 0 through 15 giving the destination of each item. The input matrix I and the output matrix O are

$$I = \begin{pmatrix} 1 & 13 & 2 & 6 \\ 9 & 0 & 15 & 12 \\ 10 & 11 & 8 & 3 \\ 5 & 7 & 14 & 4 \end{pmatrix} \quad O = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{pmatrix}.$$

The required matrix permutation can be decomposed as follows, where A_1 and A_3 are within-rows-permutations and A_2 is a within-columns-permutation:

$$A_1 \circ I = \begin{pmatrix} 6 & 13 & 1 & 2 \\ 9 & 0 & 12 & 15 \\ 3 & 8 & 10 & 11 \\ 14 & 7 & 5 & 4 \end{pmatrix}$$

$$A_2 \circ A_1 \circ I = \begin{pmatrix} 3 & 0 & 1 & 2 \\ 6 & 7 & 5 & 4 \\ 9 & 8 & 10 & 11 \\ 14 & 13 & 12 & 15 \end{pmatrix}$$

$$A_3 \circ A_2 \circ A_1 \circ I = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{pmatrix} = O.$$

C. MIN Implementation

In this subsection we will describe a general architecture that provides a bandwidth-efficient implementation for any homogeneous trellis. Given the trellis, first select any matrix permutation for the trellis. This matrix permutation will permute the items in a Pd-by-(N/P) matrix. Next, use Theorem 1 to divide the matrix permutation into a within-rows-permutation, followed by a within-columns-permutation, followed by a within-rows-permutation. This division of the matrix permutation can be found in $O(Nd \log(Nd) \log(N/P))$ time on a sequential computer or in $O(\log^2(Nd) \log(N/P))$ time on a shared memory parallel computer with Nd processors [9]. Then, use random-access memory for both banks of each ACS's local memory. This allows both within-rows-permutations to be performed by simply using the correct addressing patterns.

Finally, use a MIN to implement the within-columns-permutation. In particular, use a Pd input Benes network [2] to perform the within-columns-permutation. A Benes network with Pd inputs consists of $2\log(Pd) - 1$ stages, where each stage has $(Pd/2)$ switches (Fig. 4). Each switch takes 2 inputs and creates 2 outputs. The switch can be set to pass the inputs to the outputs either with or without swapping them. It has been shown that a Benes network can implement any desired permutation of its inputs by correctly setting its switches [2]. Thus, a Pd input Benes network can be used to implement the within-columns-permutation by shifting each column of Pd items into the network and setting the switches correctly. If it takes more than a single cycle to perform a pass through the Benes network, it can be pipelined so that multiple columns can be present in the network at a time.

Thus, the use of random-access memory and a Benes network results in a general architecture which can support any communication requirements. Of course, there is a cost associated with this generality. In particular, the use of random-access memory has two drawbacks: 1) the read and write addressing patterns for each ACS have to be stored, and 2) a multiport memory is required to read and write d metrics at a time. We will discuss local memory design in more detail in Section V, and will present a less costly implementation. Also, the Benes network is not needed in all cases. In the next section, we will consider cases in which the Benes network can be replaced by a fixed-interconnection network.

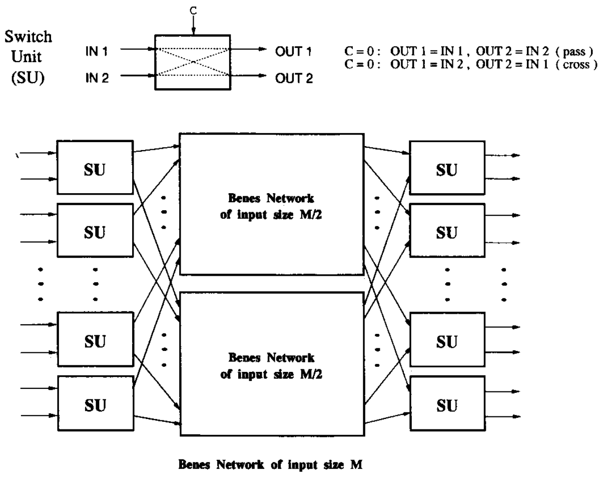

Switch Unit (SU)
C
IN 1
IN 2
OUT 1
OUT 2
C = 0 : OUT 1 = IN 1 , OUT 2 = IN 2 (pass)
C = 0 : OUT 1 = IN 2 , OUT 2 = IN 1 (cross)
SU
Benes Network of input size M/2
Benes Network of input size M/2
Benes Network of input size M

Fig. 4. The Benes multistage interconnection network (MIN). Each 2×2 switch unit (SU) can be set to *pass* or *cross*. A Benes MIN with $2 \log M - 1$ stages (each with $M/2$ SU's) can perform any permutations of size M.

D. Fixed Interconnection Implementation

In order to obtain a fixed-interconnection solution, the second permutation, A_2, must be a special type of within-columns-permutation, namely, one in which the permutation that is performed in each of the (N/P) columns is identical. In other words, the second permutation can be viewed as permuting the row vectors. Such permutations will be called *row-reordering-permutations.* We will use $\hat{\mathcal{C}}$ to denote the set of row-reordering-permutations.

Note that a similar three-part division of the matrix permutation was used in the preceding subsection in order to obtain a MIN implementation. However, in the MIN implementation, the first and third permutations were required to be within-rows-permutations. This is unnecessarily restrictive, as a block of d consecutive rows is stored locally in each ACS. As a result, we will allow the first and third permutations to be *within-blocks-permutations,* denoted by $\mathcal{B}$, where the Pd rows are viewed as forming P blocks of d rows each. This greater flexibility in selecting the first and third permutations will increase our ability to find a three-part division of the matrix permutation in which the second permutation is a row-reordering-permutation.

Unfortunately, not all matrix permutations can be divided into a within-blocks-permutation, followed by a row-reordering-permutation, followed by a within-blocks-permutation. Furthermore, the problem of deciding whether or not such a division exists can be very difficult. In other words,

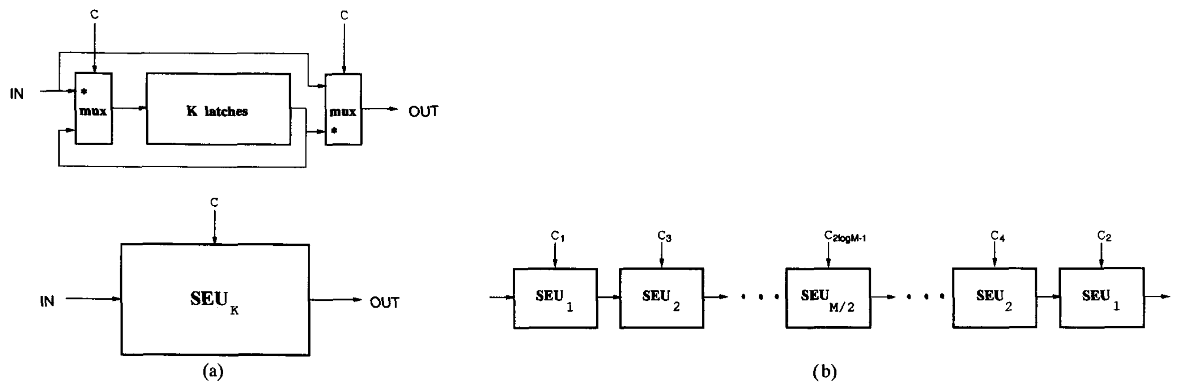


Fig. 5. (a) The shift exchange unit (SEU). A SEU of size K ($\mathbf{SEU}_K$) contains 2 multiplexers and K latches. In the *shift* mode, the two multiplexers select the input denoted by the asterisks. In the *exchange* mode, they select the other pair of inputs. $\mathbf{SEU}_K$ is capable of exchanging any pair of inputs that are separated in K stages. (b) The sequential permutation network (SPN). An SPN of size M contains $2 \log M - 1$ SEU's of size $1, 2, 4, \cdots, M/2, \cdots, 4, 2, 1$, respectively, and is capable of performing any permutation of size M in a sequential manner.

it can be very difficult to select a partitioning and scheduling matrix S in a fixed-interconnection implementation. We will now present a different approach to the problem which can provide more insight.

Definitions: Suppose $G = (V_G, E_G)$ and $H = (V_H, E_H)$ are trellises.[4] A function $f : V_H \rightarrow V_G$ is an *emulation* of H by G if, for every edge $(h_1, h_2) \in E_H, (f(h_1), f(h_2)) \in E_G$. The emulation f is *totally uniform* if, for each vertex $g \in V_G$, there are exactly $|V_H|/|V_G|$ vertices $h \in V_H$ such that $f(h) = g$, and for each edge $(g_1, g_2) \in E_G$, there are exactly $|V_H|/|V_G|$ edges $(h_1, h_2) \in E_H$ such that $f(h_1) = g_1$ and $f(h_2) = g_2$.[5]

The following theorem shows that the problem of finding a fixed-interconnection realization is equivalent to finding a homogeneous trellis with P states which can emulate the homogeneous trellis with N states in a totally uniform manner. The proof is given in the Appendix A.

Theorem 2: Let H be a homogeneous trellis with N states and degree d. Then there exists a matrix permutation $M \in \mathcal{M}$ which is valid for H and permutations A_1, A_2, and A_3 such that $A_1 \in \mathcal{B}, A_2 \in \hat{\mathcal{C}}, A_3 \in \mathcal{B}$, and $M \equiv A_3 \circ A_2 \circ A_1$, if and only if there exists a homogeneous trellis G with P states which emulates H in a totally uniform manner.

A computer search program can be constructed to find a P-state homogeneous trellis that can emulate the original N-state trellis in a totally uniform manner. For arbitrary trellises with large N, this can be very computationally expensive. For several kinds of trellises of practical interest (which will be described in Section II of the accompanying (Part II) paper), we have found the emulation solution. For small trellises, manual analysis is possible, taking advantage of any regularity or symmetry in the trellis.

In summary, fixed-interconnection implementations are desirable and possible for many practical trellises. However, the authors do not know a way to search for the solution efficiently for arbitrary trellises. Nevertheless, area-efficient implementations in a bandwidth-efficient manner using a MIN routing network can always be guaranteed, in which case an arbitrary partitioning and scheduling matrix can be used.

IV. Local Memory Design

In this section we discuss the design of the local memory. In previous sections, a multiport, random-access memory was assumed. Although conceptually simple, it is not an ideal implementation because of its complexity.

The local memory must meet a pair of conflicting goals. On the one hand, it has to be *regular* in order to allow pipelining. As will be seen in the following sections, pipelining has an important role in the area-efficient architectures. On the other hand, it has to perform the within-rows-permutation or the within-blocks-permutation. We propose a novel architecture, called the *sequential permutation network* (SPN), which can meet both goals. The idea is to allow some *reordering* in the pipeline such that the output ordering is made different from the input ordering.

In Fig. 5(a), a *shift exchange unit* (SEU) is illustrated. Note that an SEU of size K has $K+1$ pipeline registers. For each of the two pipeline stages which has two inputs (the remaining ones all have one input), a multiplexer is inserted in front. When there is no exchange, the multiplexers choose the input from their left neighbors. When there is an exchange, the multiplexers choose the other pair of inputs, thus swapping the two values in the two pipeline registers that are K stages apart. The two multiplexers are controlled by the same control signal, with which we can perform an exchange between any pair of input values that are separated by K stages.

An SPN of size M [see Fig. 5(b)] is composed of $2 \log M - 1$ SEU's of sizes $1, 2, \cdots, (M/4), (M/2), (M/4), \cdots, 2, 1$.[6] It can be shown that an SPN of size M can support any permutations of the same size. In particular, the $2 \log M - 1$ SEU's correspond to the $2 \log M - 1$ stages in a Benes network. The M values shifted out of each SEU correspond to the M outputs of the $(M/2)$ switches of the corresponding stage in

[4] Self-loops (vertices which point to themselves) and multiple edges between the same pair of vertices are allowed in both G and H.

[5] The terminology and definitions are taken from [4].

[6] Without loss of generality, we assume $M = 2^r$. If, in fact, $M \neq 2^r$, then we can replace it with M' which is the smallest integer greater than $M, M' = 2^r$.

the Benes network. Therefore, we can view the SPN as a sequential Benes network.[7]

Within-row-permutations can be implemented by SPN's directly. The input and output local memories associated with each ACS require d SPN's of size (N/P). The d SPN's of the output local memory all take as input the output of the corresponding ACS. Within-blocks-permutations, on the other hand, can be implemented by a combination of SPN's and MIN's. To see this, recall that a within-blocks-permutation is also a matrix permutation of size d by (N/P). Therefore, it can be implemented by 2 MIN's of size d and d SPN's of size (N/P).[8]

There are two other advantages to implementing a pipelined local memory with an SPN. First, the circuit between any two pipeline stages is very simple (at most a multiplexer), and hence will not constitute the timing bottleneck in the pipeline. Second, note that the signals wires do not intersect, and hence the layout of an SPN is straightforward and compact.

One drawback of the SPN is its length, or latency, which is

$$\sum_{K=1,2,4,\cdots,\frac{M}{2},\cdots,4,2,1} K+1 = \frac{3}{2}M + 2\log M - 1 \simeq \frac{3}{2}M.$$

However, for particular codes, the within-rows-permutations often can be implemented with a special-purpose SPN with a considerably smaller length. Moreover, the within-blocks-permutations can sometimes be simplified to use one SPN with multiple *tap* points. These simplifications will be shown to be effective in the accompanying paper.

V. ACS Pipelining

In the previous two sections, we have seen that the SPN and MIN can introduce significant latency. In this section, we consider the use of ACS pipelining to combat the latency problem.

Pipelining the ACS's has been proposed to increase the throughput rate by interleaving a number of independent data sources. In this case, the parallelism can be thought of as being externally created, so we will call it *external parallelism.* As mentioned earlier, when the channel has memory or intersymbol interference, the external parallelism is no longer applicable. In our architecture model, however, additional parallelism is created internally by allowing several trellis states to share an ACS. We call this *internal parallelism.* Therefore, even when the channel has memory or intersymbol interference, in an area-efficient design ACS pipelining can still be applied to take advantage of the internal parallelism. On the other hand, if the channel is memoryless, the internal parallelism can be exploited in additional to the external parallelism. ACS pipelining exploiting the internal parallelism is the key reason for the favorable area-time tradeoff in our area-efficient architectures.

The ACS for a degree-d trellis contains d adders and $d-1$ comparators and selectors. Compared to the multiplexers in an SPN or the switches in the MIN, the ACS is more complex and hence can accommodate more pipeline stages. Even though adding pipeline stages in the ACS increases the total latency in the feedback loop, this effect is *additive,* while the speedup effect by pipelining is *multiplicative.* Therefore, the overall effect of ACS pipelining is positive. We will show in the next section that it is advantageous to have as many ACS pipeline stages as possible. However, we cannot put too many such that the pipeline delays within the ACS are smaller than those in the SPN and the MIN. Consequently, the optimal ACS pipelining is achieved when the pipeline delays within the ACS are equal to those in the SPN and MIN.

VI. Area-Time Analysis

With (N/P) states sharing one ACS, a *linear scale solution* demands that the speed of the area-efficient architecture be (N/P) times slower. In this section, we will show that, with the help of ACS pipelining, our area-efficient architecture can achieve a throughput rate which is higher than that of a linear scale solution. In other words, the area-efficient technique not only allows the exploration of the design space by trading speed for area saving, but also achieves a smaller area-time product than the state-parallel implementation.

Let α be the number of pipeline stages in the ACS, β be the total number of pipeline stages in the SPN and MIN, and γ be a *overhead factor.* The condition for our area-efficient architecture to perform better than the linear scale solution is

$$\frac{\alpha+\beta}{\alpha} < \frac{1}{1+\gamma}\frac{N}{P}. \tag{1}$$

The numerator on the left-hand side of (1) indicates the total number of pipeline stages (latency) in the feedback loop. The left-hand side of (1) indicates the effective slow-down of the area-efficient architecture, compared to a nonpipelined state-parallel implementation. The right-hand side of (1) indicates the effective area saving. Equation (1) means that if the effective slow-down is smaller than the effective area saving, then the area-time tradeoff is favorable for the area-efficient architecture.

Both β and γ are trellis dependent. Let $\beta = \beta_1 + \beta_2$, where β_1 is the number of pipeline stages in the SPN and β_2 is the number of pipeline stages in the MIN. For arbitrary codes $\beta_1 = 3(N/P), \beta_2 = 2\log Pd - 1$. For specific codes, however, the routing latency can be substantially smaller with special-purpose SPN's. The γ factor is to account for the overhead related to the routing network, pipeline registers, etc., which depends strongly on the particular codes. Note that the reduced number of interconnection wires (bandwidth-efficiency) contributes to the area saving and also partially compensates the overhead.

Note that $\alpha+\beta \geq (N/P)$ because each ACS needs at least (N/P) stages to store the path metrics of all the states sharing it. The excessive pipeline stages (of size $\alpha+\beta-(N/P)$) are so-called *pipeline bubbles* which render the architecture

[7]We choose not to call an SPN a sequential Benes network because 1) at each stage the SEU can manipulate a value more than once due to its sequential nature, and 2) the architecture of an SPN can support other types of MIN's [14].

[8]From Theorem 1 it is obvious that there exists another decomposition in which A_1 and A_3 are within-columns-permutations, and A_2 is a within-rows-permutation. This decomposition is less costly for the within-blocks-permutation because $d < (N/P)$.

time-inefficient.[9] Fortunately, the internal parallelism can be exploited to speedup the architecture by a factor of α. From the above equation, it is clear that the ACS pipelining has an additive effect on the total latency and a multiplicative effect on the throughput. Hence, it is advantageous to have α as large as possible. On the other hand, as explained in the previous section, α is also constrained by the fact that the pipeline delays in the ACS cannot be smaller than those in SPN and MIN. Therefore, the optimum α (denoted by $\overline{\alpha}$) occurs when the pipeline delay in the ACS is the same as the rest of the architecture (SPN and MIN).

VII. Conclusion

In this paper, we propose a new class of area-efficient architectures for the Viterbi algorithm that allows a number of trellis states to share one ACS. A systematic approach for partitioning, scheduling, and mapping the N trellis states to P ACS's is presented. Based on the matrix permutation techniques, the partition and schedule of the trellis can be mapped to an area-efficient architecture with a multistage interconnection network (MIN). For trellises where emulation by a smaller trellis is possible, further area saving can be achieved with a fixed-interconnection realization.

We proposed a novel technique for path metric storage, by using a sequential permutation network (SPN) to reorder the path metrics. This technique results in a pipelined local memory which is faster and smaller than a multiport implementation.

A major advantage of our area-efficient architectures is the *internal parallelism,* which is created by ACS sharing. Pipelining the SPN and MIN introduces latency to the architecture, but pipelining the ACS is found to be an effective way to combat the latency problem. Detailed area-time analysis of our architectures is provided.

The favorable area-time tradeoff will be demonstrated by application examples in an accompanying paper. These include convolutional codes, matched-spectral-null (MSN) codes, and Ungerboeck codes. Applications to trellises with very large numbers of states and time-varying codes will also be discussed. The area-efficient architecture also makes it possible to design a programmable Viterbi decoder due to the extensive amount of programmability in the SPN and MIN.

Appendix A
Proof of Theorem 2

Theorem 2: Let H be a homogeneous trellis with N states and degree d. Then there exists a matrix permutation $M \in \mathcal{M}$ which is valid for H, and permutations A_1, A_2, and A_3 such that $A_1 \in \mathcal{B}, A_2 \in \hat{\mathcal{C}}, A_3 \in \mathcal{B}$, and $M \equiv A_3 \circ A_2 \circ A_1$, if and only if there exists a homogeneous trellis G with P states which emulates H in a totally uniform manner.

Proof: First we will assume that G exists, and we will show that suitable permutations A_1, A_2, and A_3 must exist. Number the vertices in G with 0 through $P-1$. Because G emulates H in a totally uniform manner, there exists a mapping f which sends (N/P) vertices in H to each vertex in G, and (N/P) edges in H to each edge in G. We will use this mapping function f to create the matrix permutations M, A_1, A_2, and A_3.

Label each vertex in H with a pair of the form $[i,j], 0 \leq i < P, 0 \leq j < (N/P)$, indicating that it is the jth vertex in H which f maps to vertex i in G. Define a partitioning and scheduling matrix S (see Section III-A), such that $s_{ij} = [i,j]$ for all i and j, $0 \leq i < P, 0 \leq i < (N/P)$. Create the matrix X by forming d copies of each row in S (see Section III-A). Thus, for all i and j, $0 \leq i < Pd, 0 \leq j < (N/P)$, $x_{ij} = [\lfloor i/d \rfloor, j]$. Create the matrix Y by replacing each entry s_{ij} with a column of d entries giving the d old path metrics which are needed in order to calculate s_{ij} (see Section III-A). Specifically, for all i and j, $0 \leq i < Pd, 0 \leq j < (N/P)$, $y_{ij} = [a,b]$ if the (i mod d)th incoming edge to node $[\lfloor i/d \rfloor, j]$ in H comes from node $[a,b]$ in H. Finally, let M be a matrix permutation defined by X and Y.

We will now show how M can be decomposed into permutations A_1, A_2, and A_3 of the desired types. Label each edge in H with a triple of the form $\langle i,j,k \rangle$, $0 \leq i < P, 0 \leq j < d, 0 \leq k < (N/P)$, indicating that it is the kth edge in H which f maps to the jth outgoing edge from node i in G. Note that the outgoing edges leaving any node $[a,b]$ in H have labels of the form $\langle a,j,k \rangle$, because f maps the node $[a,b]$ to node a in G, and f maps all of the edges which leave $[a,b]$ to edges which leave a. Permutation A_1 is defined to send each item x_{rc}, $0 \leq r < Pd, 0 \leq c < (N/P)$, to row $id+j$ and column k if the (r mod d)th outgoing edge of node $[\lfloor r/d \rfloor, c]$ is labeled $\langle i,j,k \rangle$. It was just shown that $\lfloor r/d \rfloor$ must equal i, so each item will remain within the same block of d rows, and $A_1 \in \mathcal{B}$.

Permutation A_2 permutes the rows according to the pattern of connections in G. More formally, permutation A_2 sends each item in row r and column c to row $id+j$ and column c if the (r mod d)th outgoing edge from node $\lfloor r/d \rfloor$ in G is also the jth incoming edge to node i in G. It is clear that A_2 keeps the same set of items in each column, and that items which start in the same row are sent by A_2 to the same destination row, so $A_2 \in \hat{\mathcal{C}}$.

The definition of permutation A_3 is similar to the definition of permutation A_1, except that it is stated in terms of incoming edges rather than outgoing edges. Label each edge in H with a triple of the form (i,j,k), $0 \leq i < P, 0 \leq j < d, 0 \leq k < (N/P)$, indicating that it is the kth edge in H which f maps to the jth incoming edge to node i in G. Note that the incoming edges entering any node $[a,b]$ in H have labels of the form (a,j,k), because f maps the node $[a,b]$ to node a in G, and f maps all of the edges which enter $[a,b]$ to edges which enter a. Permutation A_3 is defined to send each item from row r and column c, $0 \leq r < Pd, 0 \leq c < (N/P)$, to row $id+j$ and column k it the jth incoming edge to node $[i,k]$ is labeled (a,b,c), where $a = \lfloor r/d \rfloor$ and $b = r$ mod d. It was just shown that $\lfloor r/d \rfloor$ must equal i, so each item will remain within the same block of d rows, and $A_3 \in \mathcal{B}$.

It has been shown that $A_1 \in \mathcal{B}, A_2 \in \hat{\mathcal{C}}$, and $A_3 \in \mathcal{B}$, but it still remains to be shown that $M \equiv A_3 \circ A_2 \circ A_1$. Consider an arbitrary item in row $id+j$ and column k of

[9] Of course, if external parallelism is applicable, then the pipeline bubbles can be eliminated, and the area-efficient architecture will be time-efficient.

X, where $0 \leq i < P, 0 \leq j < d$, and $0 \leq k < (N/P)$. From the definition of X, this item equals $[i, k]$. Following the application of permutation A_1, this item is located in row $id+a$ and column b if the jth outgoing edge leaving node $[i, k]$ is labeled $\langle i, a, b\rangle$. Then, following the application of permutation A_2, this item is located in row $gd + h$ and column b if the ath outgoing edge leaving node i in G is also the hth incoming edge entering node g in G. Finally, following the application of permutation A_3, this item is located in row $gd + u$ and column v if the uth incoming edge entering node $[g, v]$ is labeled (g, h, b). Note that the jth outgoing edge leaving node $[i, k]$ is the bth edge which is mapped by f to the ath outgoing edge leaving node i in G. Also, note that the uth incoming edge entering node $[g, v]$ is the bth edge, which is mapped by f to the hth incoming edge entering node g in G. But the ath outgoing edge leaving node i in G is also the hth incoming edge entering node g in G. Therefore, the jth outgoing edge leaving node $[i, k]$ is also the uth incoming edge entering node $[g, v]$. From the definition of Y, the item in row $gd + u$ and column v of Y has the value $[i, k]$. It was just shown applying permutations A_1, A_2, and A_3 to X, the item in row $gd+u$ and column v has the value $[i, k]$. As a result, $M \equiv A_3 \circ A_2 \circ A_1$.

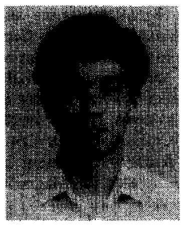
Now we will assume that a matrix permutation M and suitable permutations A_1, A_2, and A_3 exist, and we will show that a homogeneous trellis G with P states which emulates H in a totally uniform manner must exist. The construction is essentially the inverse of the previously described construction. Let matrices S, X, and Y be as defined in Section III-A, such that the matrix permutation M maps matrix X into matrix Y. Number the vertices in G with 0 through $P-1$. Each vertex in G will have exactly d outgoing and d incoming edges. Assign to the ith outgoing edge, $0 \leq i < d$, from vertex j in G the number $jd + i$. Construct G by making each outgoing edge number k, $0 \leq k < Pd$, the ith incoming edge to vertex j if permutation A_2 sends row k to row $jd+i$. It is easily verified that G emulates H in a totally uniform manner when vertex i in H is mapped to vertex j in G and only if there exists an entry s_{rc} in S such that $s_{rc} = i$ and $r = j$. □

ACKNOWLEDGMENT

The authors wish to thank R. Brodersen and J. Hwang for many helpful discussions.

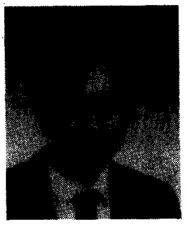
REFERENCES

[1] V. E. Benes, "On rearrangeable three-stage connecting networks," *Bell Syst. Tech. J.*, vol. 41, pp. 1481–1492, 1962.
[2] ——, "Optimal rearrangeable multistage connecting networks," *Bell Syst. Tech. J.*, vol. 43, pp. 1641–1656, 1964.
[3] G. Fettweis and H. Meyr, "Parallel Viterbi algorithm implementation: Breaking the ACS-bottleneck," *IEEE Trans. Commun.*, vol. COM-37, pp. 785–790, Aug. 1989.
[4] J. P. Fishburn and R. A. Finkel, "Quotient networks," *IEEE Trans. Computers*, vol. C-31, pp. 288–295, Apr. 1982.
[5] P. G. Gulak and T. Kailath, "Locally connected VLSI architecture for the Viterbi algorithm," *IEEE J. Select. Areas Commun.*, vol. SAC-6, pp. 527–537, Apr. 1988.
[6] P. G. Gulak and E. Shwedyk, "VLSI structures for Viterbi receivers: Part I—general theory and applications," *IEEE J. Select. Areas Commun.*, vol. SAC-4, pp. 142–154, Jan. 1986.
[7] G. D. Forney, Jr., "Maximum-likehood sequence estimation of digital sequences in the presence of intersymbol interference," *IEEE Trans. Inform. Theory*, vol. IT-18, pp. 363–378, May 1972.
[8] ——, "The Viterbi algorithm," *Proc. IEEE*, vol. 61, pp. 268–278, Mar. 1973.
[9] G. F. Lev, N. Pippenger, and L. G. Valiant, "A fast parallel algorithm for routing in permutation networks," *IEEE Trans. Computers*, vol. C-30, pp. 93–100, Feb. 1981.
[10] H.-D. Lin and D. G. Messerschmitt, "Algorithms and architectures for concurrent Viterbi decoding," in *Proc. Int. Conf. Commun.*, Boston, MA, June 1989, pp. 836–840.
[11] J. Rabaey, T. Stoelzle, D. Chen, S. Narayanaswamy, R. Brodersen, H. Murveit, and A. Santos, "A large vocabulary real time continuous speech recognition system," in *VLSI Signal Processing, III*, R. Brodersen and H. Moscovitz, Ed. New York: IEEE Press, 1988.
[12] H. Thaper and J. Cioffi, "A block processing method for designing high-speed Viterbi detectors," in *Proc. Int. Conf. Commun.*, Boston, MA, June 1989.
[13] A. J. Viterbi, "Error bounds for convolutional codes and asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 260–269, Apr. 1967.
[14] C.-L. Wu and T.-Y. Feng, "On a class of multistage interconnection networks," *IEEE Trans. Computers*, vol. C-29, pp. 108–116, Aug. 1980.



C. Bernard Shung received the B.S. degree in electrical engineering from National Taiwan University in 1981, and the M.S. and Ph.D. degrees in electrical engineering from the University of California at Berkeley in 1985 and 1988, respectively.

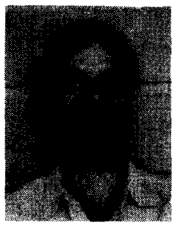
From 1988 to 1990 he was a Visiting Scientist at IBM Research Division, Almaden Research Center, San Jose, CA, where he was engaged in the development of prototyping integrated circuits for pattern recognition and magnetic recording. In 1990 he joined the Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, Republic of China, where he is now an Associate Professor. His research interests include VLSI circuits and systems for communications and signal processing, and computer-aided design for integrated circuits.



Horng-Dar Lin received the B.S.E.E. degree from the National Taiwan University in 1984, and the M.S. and Ph.D. degrees from the University of California at Berkeley in 1988 and 1991, respectively.

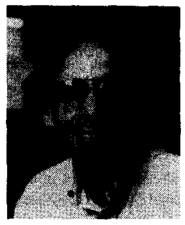
Between 1984 and 1986 he served as a Technical Officer in Taiwan responsible for research and engineering in communication electronics. He worked on efficient ASIC pipelining at the IBM Almaden Research Center between 1989 and 1990, and on optical networks at the IBM T.J. Watson Research Center in 1990. Between 1990 and 1991 he worked as the principle architect of an image compression encoder/decoder chip at Teknekron Communications Inc., Berkeley, CA. Since 1991 he has been with the AT&T Bell Labs and continued his VLSI research. His interests include algorithm-to-architecture mapping, power-efficient circuits, parallel algorithms, application-specific memory structures, and high-speed communications and signal processing.

Dr. Lin was elected a member of the Phi Tau Phi Scholastic Honor Society in Taiwan in 1984.



Robert Cypher was born in Schenectady, NY, in 1959. He received the B.S. degree in mathematical sciences from Stanford University in 1982 and the M.S. and Ph.D. degrees in computer science from the University of Washington in 1987 and 1989, respectively.

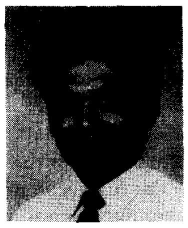
He is currently a Research Staff Member of the IBM Almaden Research Center and a Consulting Assistant Professor in the Stanford University Computer Science Department. He is interested in both the theoretical and practical aspects of parallel processing. He has done research in parallel algorithms for image processing, computational geometry, sorting, and data routing. He is also interested in VLSI, signal processing, fault-tolerance, and the design of interconnection networks.



Paul H. Siegel was born in Berkeley, CA, in 1953. He received the B.S. degree in mathematics in 1975 and the Ph.D. degree in mathematics in 1979, both from the Massachusetts Institute of Technology. He held a Chaim Weizmann fellowship during a year of postdoctoral study at the Courant Institute, New York University.

He joined the Research Staff at IBM in 1980. He is currently Manager of the Signal Processing and Coding Project at the IBM Almaden Research Center in San Jose, CA. His primary research interest in the mathematical foundations of signal processing and coding, especially as applicable to digital data storage channels. He holds several patents in the area of coding and detection for digital recording systems. He has taught courses in information and coding at the University of California, Santa Cruz, and at Santa Clara University, and was a Visiting Associate Professor at the University of California, San Diego, while at the Center for Magnetic Recording Research during the 1989–1990 academic year.

Dr. Siegel was elected to Phi Beta Kappa in 1974. He is currently a member of the Board of Governors of the IEEE Information Theory Society. He was a Co-Guest Editor of the May 1991 Special Issue on Coding for Storage Devices of the IEEE TRANSACTIONS ON INFORMATION THEORY.



Hemant K. Thapar received the Ph.D. degree in electrical engineering from Purdue University, West Lafayette, IN, in August 1979.

He worked at the Bell Telephone Laboratories, Holmdel, NJ, from October 1979 to September 1984, first on the design method for AT&T's Dynamic Non-Hierarchical Routing Network, and subsequently on signal processing and coding methods for high-speed data transmission. Since 1984 he has been with IBM, San Jose, CA, where he is a Manager in the Advanced Magnetic Recording Laboratory. He is also an Adjunct Lecturer at Santa Clara University, where he regularly teaches courses in communication theory, digital communication, and signal processing. His primary interests are in communication signal processing and VLSI design.

Dr. Thapar is a member of Tau Beta Pi and Eta Kappa Nu. He is the recipient of the Interface 1984 Best Paper Award and the IEEE Communications Society's 1991 COMMUNICATION MAGAZINE Prize Paper Award.