IEICE TRANS. COMMUN., VOL. E76-B, NO. 6 JUNE 1993

PAPER

Parallel Viterbi Decoding Implementation by Multi-Microprocessors

Hui ZHAO†, Xiaokang YUAN††, *Nonmembers*, Toru SATO† *and* Iwane KIMURA†, *Members*

SUMMARY The Viterbi algorithm is a well-established technique for channel and source decoding in high performance digital communication systems. However, excessive time consumption makes it difficult to design an efficient high-speed decoder for practical application. This paper describes the implementation of parallel Viterbi algorithm by multi-microprocessors. Internal computations are performed in a parallel fashion. The use of microprocessors allows low-cost implementation with moderate complexity. The software and hardware implementations of the Viterbi algorithm on parallel multi-microprocessors for real-time decoding are presented. The implemented method is based on a combination of forming a set of tables and calculations. For efficient operation under fully parallel Viterbi decoding by microprocessors, we considered: (1) branch metrics processing, path metrics updating, path memory updating and decoding output for microprocessor, (2) efficient decomposition of the sequential Viterbi algorithm into parallel algorithms, (3) minimization of the communication among the microprocessors. The practical solutions for the problems of synchronization among the miroprocessors, interconnection network for communication among the mircroprocessors and memory management are discussed. Further more the performance and the speed of the parallel Viterbi decoding are given. For a fixed processing speed of given hardwares, parallel Viterbi decoding allows a linear speed up in the throughput rate with a linear increase in hardware complexity.

key words: *coding, error correction code, convolutional code, Viterbi decoding, parallel algorithm, parallel Viterbi decoding, coding by microprocessor*

1. Introduction

The Viterbi Algorithm (VA) is widely used for decoding convolutional codes.[1] Although it is an optimum decoding algorithm, its computational complexity grows exponentially with the constraint length of the encoder. To enhance the achievable throughput rate of an implementation algorithm, parallel and/or pipelined architectures are used. The fully parallel Viterbi decoding algorithm was first proposed by Forney.[2] The main stumbling block for implementing parallel Viterbi decoding is its complexity in synchronization and contention among the processors and occupation of large space by connection lines. Usually pipelined algorithms provide inferior throughput but fewer timing problems, so it is a trade-off between the throughput rate and the implementation complexity. For high-speed implementation of an algorithm, architectures are desired that lead to a linear increase in hardware complexity for a linear speedup in the throughput rate if the limit of computational speed of the hardware is reached. Much research has been done in recent years[3]–[7] on VLSI implementation of parallel/pipelined VA. Some research aimed at microprocessor implementation for VA are found.[8]–[10] While Viterbi decoding by two microprocessors is implemented by Zhao,[11] practical implementation of parallel VA by microprocessors has not yet been made.

In this paper a fully parallel implementation of VA by microprocessors is proposed where one state is assigned for a microprocessor. A convolutional code of constraint length K has 2^K states, and hence, 2^K microprocessors. For values of K greater than 8, Viterbi decoding becomes increasing complex and impractical. On the other hand, the required hardware logic speed is only $1/2^K$ operations per symbol interval. This type of full parallel layout, though dominated by a large interprocessor wire area, is the architectural organization with the highest possible throughput for a given microprocessor.

Viterbi decoding by a microprocessor is briefly described in Sect. 2, emphasizing on branch metrics processing, path metrics updating, path memory updating and decoding output for microprocessor. A fully parallel VA for multi-microprocessors is introduced in Sect. 3, where details are given on decomposition of the sequential VA into parallel VA and problems to be solved in the parallel Viterbi decoding by microprocessors. Hardware implementation of parallel Viterbi decoding is described in Sect. 4, where practical implementation of synchronization among the microprocessors, interconnection network for communication among the microprocessors and memory management are discussed. Section 5 describes the results.

2. Viterbi Decoding by a Microprocessor

The VA is mainly used for decoding convolutional codes.[1],[2],[12]–[15] In general, the Viterbi

Manuscript received October 6, 1992.

† The authors are with the Faculty of Engineering, Kyoto University, Kyoto-fu, 606-1 Japan.

†† The author is with Shanghai Research Institute of Radio Equipment, Ministry of Aeronautic and Astronautic Industry, Shanghai 200082, P. R. China.

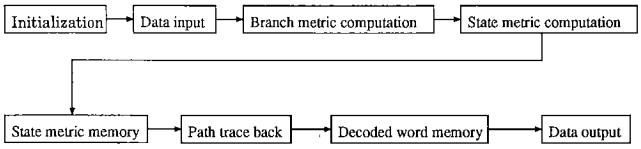


Fig. 1 General serial architecture of a Viterbi decoder.

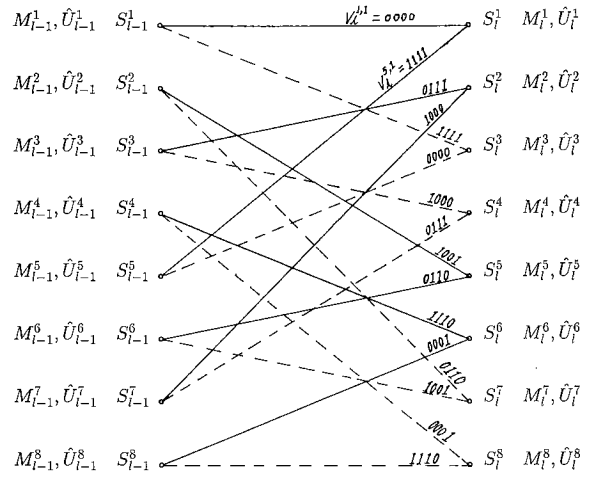


Fig. 2 The l-th stage trellis diagram of (4, 1, 3) convolutional code.

decoding process can be divided into three stages: branch metric generation, path metric updating and path memory(survivor) updating. In this section we describe VA by a microprocessor and propose new implementation methods on branch metric generation, path metric updating and path memory updating. The proposed methods differ from other schemes[8]-[10] in that they are based on a combination of forming a set of tables and calculations, and thus can reduce the execution time. Let us consider the Viterbi decoding by a microprocessor for decoding (n, k, m) convolutional code where m is the memory (constraint length$=m+1$) and k/n is the code rate. Figure 1 shows the block diagram of a serial Viterbi decoder.

2.1 Branch Metric Generation

The coded information is inputted by SIO and/or PIO interfaces, and then branch metric calculations are carried out. Figure 2 shows the l-th stage trellis diagram of the (4, 1, 3) convolutional code. The generation polynomial of the code is given by

$$\boldsymbol{G}=[1+D^2+D^3,\ 1+D+D^3,\ 1+D+D^3,\ 1+D+D^2+D^3],$$

where S_l^i, M_l^i and $\hat{U}_l^i(i=1,\cdots,8,\ l=1,\cdots,L)$ denote the i-th state of the l-th stages, the path metrics of S_l^i and the decoded information of S_l^i, respectively. In the diagram, the solid lines mean input branch signal $\boldsymbol{u}_l=0$, and the dashed lines mean input branch signal $\boldsymbol{u}_l=1$. The 4-digit numbers over the solid and dashed lines indicate branch codeword $\boldsymbol{v}_l=v_l^{(1)}v_l^{(2)}v_l^{(3)}v_l^{(4)}$. When decoding by a microprocessor, the branch metrics are calculated for all possible branches at every stage $l(m<l<L)$. According to the l-th received branch $\boldsymbol{r}_l$ (codeword), the Viterbi decoder computes the branch metrics merging into a state $S_l^i(i=1,\cdots,8,\ l=1,\cdots,L)$:

$$\Delta M_l^{j,i}=d(\boldsymbol{r}_l,\ \boldsymbol{v}_l^{j,i}), \tag{1}$$

where $\Delta M_l^{j,i}$ represents the branch metrics from state j (at stage $l-1$) to state i (at stage l), and $d(\boldsymbol{r}_l,\ \boldsymbol{v}_l^{j,i})$ denotes the Hamming distance of $\boldsymbol{r}_l$ and $\boldsymbol{v}_l^{j,i}$. Since there are two branches merging into a state, there are two branch metrics $\Delta M_l^{j,i}$ to the state. For Viterbi decoding by a microprocessor the calculation of $d(\boldsymbol{r}_l,\ \boldsymbol{v}_l^{j,i})$ can be simplified as the form of looking up a branch metric table. Since $v_l^{j,i}$ is a known codeword for certain i, j and l, we can makeup a table for branch metrics as follows: When an r_l is received, it is used as the lowest four bits of the beginning address of the branch metric table, and $\Delta M_l^{j,i}$ can be obtained by the table. The speed of looking up a table is faster than calculating the branch metric each time. For example, for the state 1, there are two branches merging in, $v_l^{1,1}=1100$ and $v_l^{5,1}=1111$. If $r_l=0000$, then $\Delta M_l^{1,1}=2$, $\Delta M_l^{5,1}=4$ and $\Delta M_l^{j,i}=+\infty$ for $j\neq 1, 5$. In the implementation, we only need to calculate two branch metrics which merges to a state. Other branches do not have really connection with this state, and we do not need to calculate the branch metrics on these branches. One state needs 32-memory units for the branch metric table. For different r_l we can get different $\Delta M_l^{j,i}$ according to the branch metric table.

2.2 Path Metric Updating

The branch metrics are used in conjunction with the stored state metrics at stage $l-1$ to compute the path metrics of the present stage l. It reads out an old state metric value stored in a RAM, adds the corresponding branch metric, and gets the path metric

$$M_l^{j,i}=M_{l-1}^j+\Delta M_l^{j,i}. \tag{2}$$

In a binary symmetric channel, the minimum Hamming distance of a code is considered to be the criteria of the maximum likelihood decoding. The Viterbi decoder compares two path metrics leading to a state, and chooses the path metric which has the minimum value as the state metric of the state

$$M_l^i=\min_{1\leq j\leq 8}[M_l^{j,i}]. \tag{3}$$

2. 3 Path Memory Updating

The Viterbi decoder selects a path which has the minimum metric as the retained path of every state. Of cause Eq. (3) is just the updating of the path metric. The state $\hat{j}$, from which the path of the minimum metric (the most likely path) comes from, is given by

$$\hat{j}=\min_{1\leq j\leq 8}{}^{-1}[M_l^{j,i}], \tag{4}$$

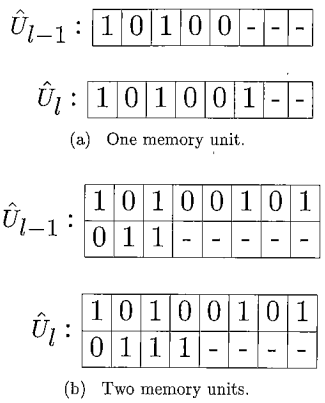
where $\min^{-1}$ means to take the j that yields the minimum value of $M_l^{j,i}$. Equation (4) is used to keep track of the most likely path. For any convolutional code, some $\Delta M_l^{j,i}$ may not exist. This problem can be usually solved by making $\Delta M_l^{j,i}$ a very large number so that this branch will never be chosen, but we can simplify this procedure. Since in fact only two branches marge into a state, Viterbi decoder only need to process two paths merging into a state.

According to the retained path, the Viterbi decoder gets the decoding information,

$$\hat{U}_l^i=\hat{U}_{l-1}^{\hat{j}}+\Delta\hat{U}_l^{\hat{j},i}, \tag{5}$$

where $+$ means bit concatenation, $\hat{U}_{l-1}^{\hat{j}}$ is the estimated decoded information of $(l-1)$-th stage, and $\Delta\hat{U}_l^{\hat{j},i}$ is the decoded branch information bits at $\hat{j}\rightarrow i$ branch in stage l. In order to obtain the minimum error rate, the length of the information sequence updating should be made as long as possible. The difficulty with a long survivor path is the decoding delay and storage needed for storing a long path plus its metric. The compromise is to truncate the path memory of the decoder by storing only the latest τ blocks of information bits for each survivor. The choice of τ involves a trade-off between two conflicting factors: the decoding accuracy and the decoding difficulty. Experiences and analyses have shown[12] that if τ is on the order of five times the encoder memory or more, all 2^K survivors stem from the same information block τ time steps back, with probability approaching 1, and there is no ambiguity in making the decoding decision. For (4, 1, 3) convolutional code $\Delta\hat{U}_l^{j,i}$ is a one bit information, and $\tau=5m=15$ is selected as the length of the survivor path memory. In practical cases a scheme is used in which the first 8 bits are sent out in parallel after decoding 24 bits of information. The delay varies randomly from 16 bits to 24 bits. $\hat{U}_l^i$ will occupy three memory units (one memory unit is 8 bits). The method of management of the decoded information bits is shown below. When $\hat{U}_l^i$ is less than 8 bits, only one memory unit is used. The process of Eq. (5) is finished by putting $\hat{U}_{l-1}^{\hat{j}}$ in the left-most side of a memory unit and concatenating one bit $\Delta\hat{U}_l^{\hat{j},i}$ in right side next to the $\hat{U}_{l-1}^{\hat{j}}$. For example, if $\hat{U}_{l-1}^{\hat{j}}=$ 10100 and $\Delta\hat{U}_l^{\hat{j},i}=1$, then $\hat{U}_l^i=101001$ as shown in the upper panel of Fig. 3.

When the length of $\hat{U}_l^i$ exceeds 8 bits, another memory unit is concatenated. For example, $\hat{U}_{l-1}^{\hat{j}}=$ 10100101011 and $\Delta\hat{U}_l^{\hat{j},i}=1$, then $\hat{U}_l^i(=101001010111)$ is expressed as shown in the lower panel of Fig. 3. This method saves the CPU time because at any time only one bit needs to be inserted in one memory unit and not shift operation is required as conventional schemes do.



(a) One memory unit.

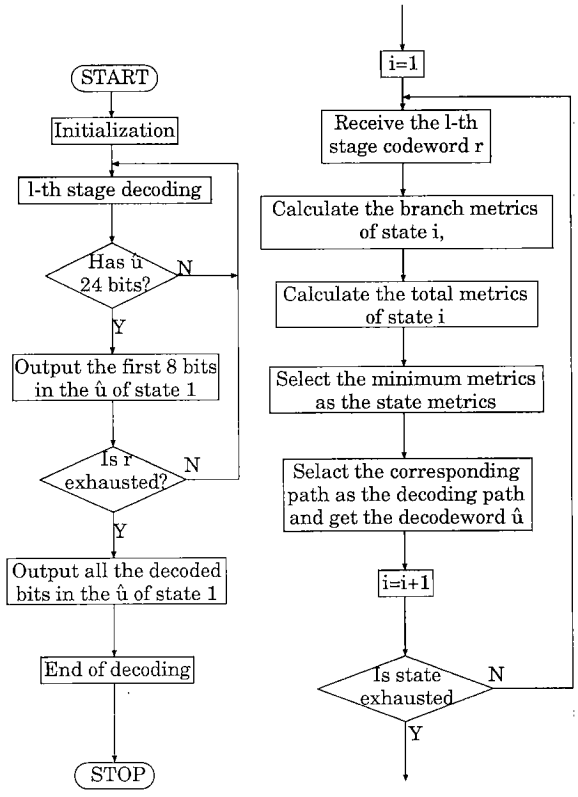
(b) Two memory units.

Fig. 3 Decoded information arrangement in a memory.

2. 4 Decoding Output

When $\hat{U}_l^i$ reaches to 24 bits, a decoding decision must be made on the first memory unit of 8 information bits. There are several possible strategies for making this decision.[12] Here we select an arbitrary survivor, and the first 8 bits on this path are chosen as the decoded bits. After the first decoding decision is made, additional decoding decisions are made in the same way for each new received block processed. Hence, the decoding decisions are always delayed from decoder input by an amount equal to the path memory size (i.e., τ blocks). The delay varies randomly from 16 bits to 24 bits.

In practical schemes for the implementation of Viterbi decoder, it is apparent that the description of the Viterbi decoding algorithm cannot be used as a practical starting point for any implementation. As a consequence, we will first propose a major simplification of the algorithm, and furthermore, introduce a new and formal notation suitable to efficiently describe the search procedure on the trellis. At the first stage, only state 1 has two branch leading to stage 2. We can assume that the state metric of state 1 at 1st stage is zero ($M_1^0=0$), and that state metrics of other states at 1st stage are sufficiently large positive value. So these paths will never be selected as the survivor path except for the path coming from state 1. It can be seen that at stage m all the survived paths are traced back to the state 1 at stage 1. Thus this simplified procedure is equivalent to the original one.



(a) Overall flowchart. (b) The l-th stage decoding.

Fig. 4 The Viterbi decoding flowchart.

Figure 4 shows the flowchart of the proposed Viterbi decoding scheme.

3. Parallel Viterbi Algorithm for Multi-microprocessors

3.1 Principle of Parallel Viterbi Algorithm

The principal limitation on the practical application of the VA is that the complexity of decoding is proportional to 2^K, the number of encoder states. The decoder memory is proportional to 2^K. Also, since 2^K comparisons must be performed per unit time, decoding time is proportional to 2^K. This exponential dependence on K limits the use of the VA to the value of K equal to 8 or less.

The speed limitation of the VA can be alleviated by employing parallel decoding. There is a special requirement for the dynamic feedback of Viterbi decoding that the next stage calculation can only be carried out after the previous stage is finished. From this point of view, we employ the parallel processing only within a single level of trellis.

Major factors to be considered in the parallel processing system are: decomposition of the sequential VA into equal parts suitable for parallel Viterbi decoding, communication among CPUs, memory management, synchronization and contention among the CPUs. Since the number of communication steps varies with scheduling, it is very important to find optimal scheduling that minimizes the number of communication steps.

Since the 2^K comparisons in the Viterbi decoding that must be performed at each time unit are identical, 2^K identical processors can be provided to do the comparisons in parallel. Each processor computes the metric values of the 2 paths entering a state, selects the path with the minimum metric, and stores that path and its metric in the decoder's memory. This parallel implementation of the VA therefore implies a factor-of-2^K speed improvement over a standard decoder, but requires 2^K times as much hardware.

3.2 Architectures of Parallel Viterbi Decoder

As we have seen that at every time instant $l\,(m<l<L)$, 2^K states will be processed one by one. we can assign a processor for each state by parallel computation.

For high speed operation, several parallel processing architectures have been proposed.[16],[17] They include pipeline processors, array processors, and full parallel processors. There are four kinds of architectures in parallel processors: SISD (Single Instruction, Single Data), SIMD (Single Instruction, Multiple Data), MISD (Multiple Instruction, Single Data), and MIMD (Multiple Instruction, Multiple Data).

We use the MIMD microprocessors architecture here for parallel VA as illustrated in Fig. 5, which is much more effective than the other architectures, but more difficult to realize. The MIMD machine consists of M processors, M memory units, interconnection network, and M controllers. The advantage of MIMD architecture is that processors can execute different instructions at same time. Such a multiprocessor system can achieve a linear speed-up factor.

The MIMD microprocessor architectures for Viterbi decoding has the following characteristics:

1. In each step, 2^K processors are used simultaneously.
2. All processors in every stage execute different sets of instructions, because each processor is assigned a

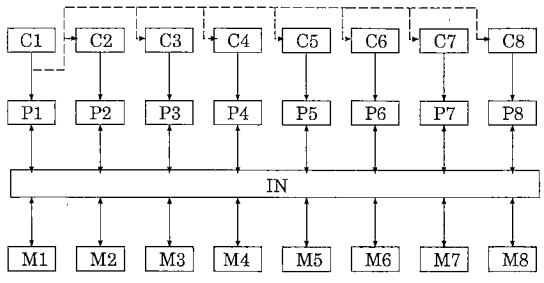


Fig. 5 MIMD microprocessors. (C: control unit, P: processor, IN: interconnection network, M: memory)

control unit.
3. The interprocessor communication between two consecutive steps is independent of the results of processing in these steps.
4. There are memory read conflicts.
5. All processors must compute synchronously at every stage.

Because the VA is related to dynamic programming and has a data-dependent feedback loop, the most difficult problems for fully parallel implementation are the timing and contention. In order to make the best use of them, waiting periods in computation should be avoided.

3. 3 Factors that Affect Decoding Speed in Parallel Algorithm

There are some cases in which parallel algorithm is not helpful in reducing execution time of an algorithm, because some processes cannot be divided into parts and cannot be assigned multiple CPUs to process them synchronously. As shown in Fig. 4, only the parts in the right panel can be processed in parallel and these parts take most of the time in Viterbi decoding (from state 1 to state 2^K). Processes shown in the left panel of Fig. 4 cannot be done in parallel algorithm. In parallel Viterbi decoding two factors affect the decoding speed: 1) Not all the processes can be done in parallel, 2) inside the parallel decoding there are also some processes which can not reduce calculating time, such as "take the received branch r_t," for which M CPUs will take exactly the same time as one CPU does. In serial decoding one CPU will take the received branch r_t once and in parallel decoding each of M CPUs will take the received branch r_t once.

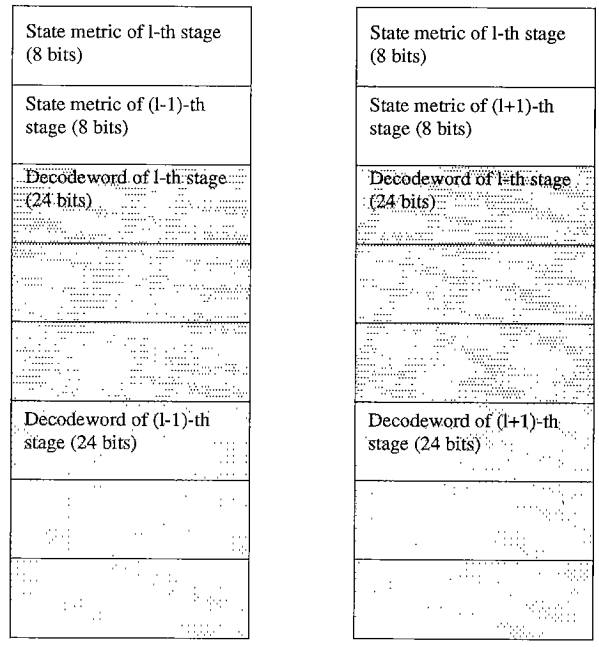
4. Technical Consideration on Hardware Implementation

4. 1 Control Units

Control units consist of EPROMs, in which management programs and Viterbi decoding programs are stored. Every processor is assigned a control unit, and instructions for the CPU are stored in the control unit. At any given time each processor can execute different instructions, and programs should be designed to let all the processors calculate synchronously in every stage.

4. 2 Memory Arrangement

For parallel Viterbi decoding, 2^K CPUs will access a memory frequently and synchronously. In this case contention will happen. In order to solve this problem two methods can be used: 1) Let CPUs access a memory one after another. In this way many wait steps occur. 2) Use separate memory blocks. Each CPU is assigned a memory block. Every CPU can access a different memory block synchronously. This second way is chosen here and 2^K separate memory blocks are used to store the information of the 2^K states. At any stage l, each memory block stores $M_l^{(i)}$, $M_{l-1}^{(i)}$, $\Delta M_{l,1}^{(i)}$, $\Delta M_{l,2}^{(i)}$, $\hat{U}_l^{(i)}$, and $\hat{U}_{l-1}^{(i)}$, $(i=1, \cdots, 2^K)$ as shown in Fig. 6, where one memory unit (8 bits) is used for each $M_l^{(i)}$, $M_{l-1}^{(i)}$, $\Delta M_{l,1}^{(i)}$ and $\Delta M_{l,2}^{(i)}$, and three memory units (24 bits) are used for each $\hat{U}_l^{(i)}$ and $\hat{U}_{l-1}^{(i)}$. When a CPU calculates at a stage, the information at the previous stage is required. In each memory block, the memory is divided into two areas for storing the information at stage $l-1$ and at stage l respectively, and these two memory areas are used alternately as shown in Fig. 6.



(a) Distribution of i-th state store area at l-th stage.

(b) Distribution of i-th state store area at (l+1)-th stage.

Fig. 6 Distribution of $S^{(i)}$ state memory area at l-th and $l+1$-th stages.

The memory unit for state metric is finite, so that overflow must be prevented. Since one memory unit (8 bits) is assigned to each state metric, the maximum value of a state metric is $2^8-1=255$. When an overflow occurs, the minimum $M_l^{(i)}$ state metric is selected as a common factor, and it is subtracted from all the state metrics producing the new state metrics.

4. 3 Interconnection Network

From Fig. 2, we can see that every CPU will access three memory blocks: two in the previous stage, and one in the present stage. Every memory block will be

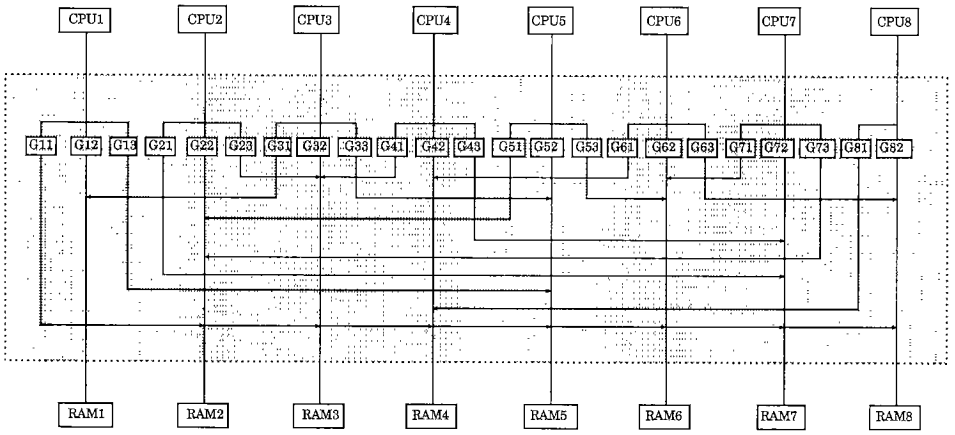


Fig. 7 Interconnection network.

accessed by three CPUs. When CPUs access the same RAM they use the same bus. So the CPUs will be connected to the common bus. As CPUs are connected to the common bus, they can use the bus only one after another. To solve this problem, an interconnection network (IN) has been designed. It has two functions: 1) data exchanges among CPUs and RAMs, and 2) preventing contention when CPUs use the same bus simultaneously. Usually two kinds of IN are used: One is using a single-set of common bus, and the other is using multi-set of common buses. The former has the advantage of simpler bus structure and hence reduces the area occupied by the buses, but if microprocessors use the buses synchronously, contention will happen. The latter will occupy a lot of area, but CPUs can use the buses synchronously and thus save the process time. Owning to the special structure of the trellis of Viterbi algorithm we propose an IN structure as shown in Fig. 7, which is different from the above schemes. It has the advantages of both of the above schemes and overcomes the disadvantages. It has the following characteristics: 1) A CPU cannot access another CPU directly, 2) A CPU can access three RAMs, but one RAM at a time, 3) The IN is controlled by CPUs.

Here we give the details of the proposed IN. It consists of connection lines and three-state control gates. One three-state control gate consists of three 74LS245 chips. One chip can control 8 lines. When a CPU wants to access a RAM block, the corresponding three-state gate is turned on and the three buses (data bus, address bus, and control bus) are connected to the RAM. Otherwise, the three-state gate is off, and the three buses of CPU cannot connect to the RAM. Three control gates are needed for each CPU, because every CPU can access three different RAM blocks. The purpose of using three-state control gates is to isolated the CPUs from each other. CPUs can access different RAM blocks at the same time and the same RAM block at different times, but cannot use the same RAM block at the same time. Our scheme on IN can save a lot of space occupied by connection lines and the control gates compared with the conventional multi-bus IN and still keeps the performance.

4.4 Synchronization of CPUs

For parallel operation of Viterbi algorithm, synchronization is essential, and a precise timing scheme is required. The following two methods are used in realizing synchronization: 1) Central processor controlling synchronization; When decoding start, the central processor sends out a signal, and all the processors start decoding synchronously. 2) Program synchronization; Because all the processors in any step execute different sets of instructions and must compute synchronously at every stage, program synchronization is used. That is to let all the processors take the same calculating time at every stage. When some processors use the same memory block at the same time, conflicts will occur. In this scheme, it is avoided by the program designs and the IN. Since the 2^K separate memory blocks are used for storing the information of 2^K states, every CPU will access three memory blocks, drawing out information from two memory blocks at $(l-1)$-th stage and storing information in one memory block at l-th stage state. We can let the CPUs to use different memory blocks every time. For CPUs of state 1 and state 3, CPU1 can first access $S_{l-1}^{(1)}$, and then $S_{l-1}^{(5)}$, and CPU3 can first access $S_{l-1}^{(5)}$ and then $S_{l-1}^{(1)}$. In this manner conflicts are avoided and the CPUs can compute continuously without waiting.

5. Experiments and Results

5.1 Experimental Setup

First, programs for serial Viterbi decoding algorithm (shown in Fig. 4) are designed and tested on a single microprocessor. The Z-80A microprocessor is

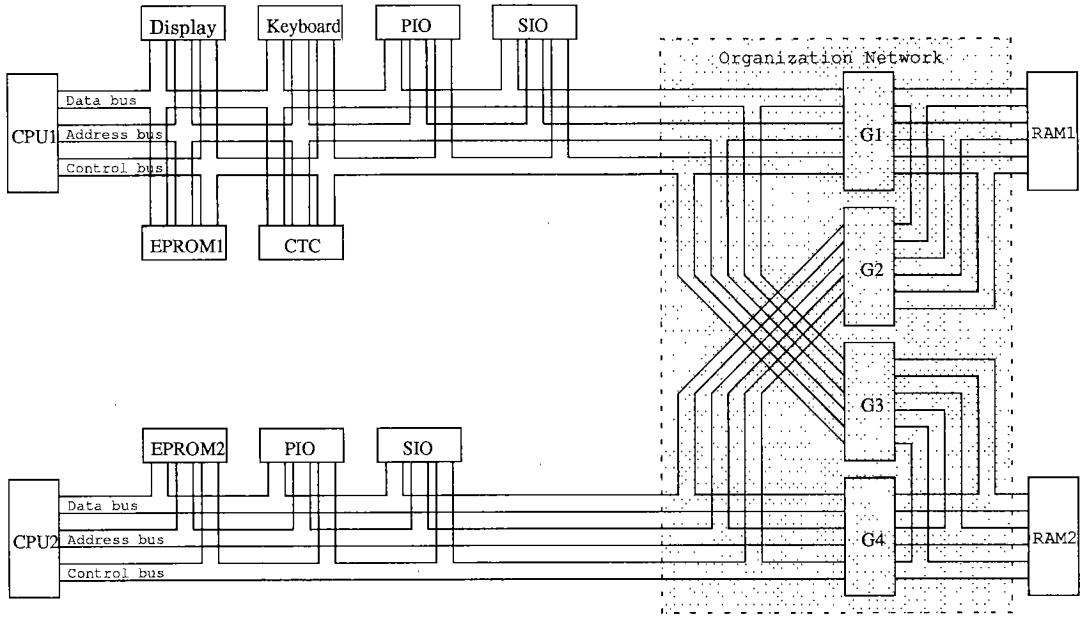


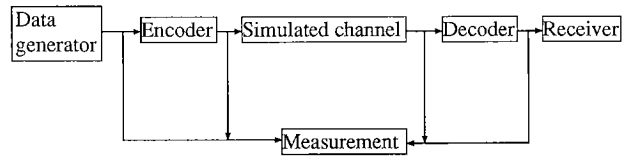
Fig. 8 Parallel Viterbi decoding by two microprocessors.

used to carry out Viterbi decoding. The programs are written by assembly mnemonic, translated into machine codes, and then stored in EPROMs. The use of assembly mnemonic has the advantages of directly using all the resources in a system, and that the exact executing time can be calculated.

Second, programs for parallel Viterbi decoding of two CPUs are designed and circuits for two-CPU parallel Viterbi decoding are designed and tested. The two-CPU parallel Viterbi decoder is based on the Z-80A microprocessor. One CPU is borrowed from the Z-80A microprocessor system including a CPU, a PIO, an SIO, a CTC, an internal clock, a power on reset circuit etc. Circuits for the other CPU system (including an EPROM, a PIO, an SIO, an interconnection network etc.) are designed. Combining these two parts by the interconnection network we get the parallel Viterbi decoder. The configuration of the parallel Viterbi decoder using two CPUs is shown in Fig. 8.

The two CPUs use the same external clock of 4 MHz. We call the CPU in the Z-80A microprocessor system as the main CPU and the other as the sub-CPU. The sub-CPU, which is equivalent to the one as used for the main CPU, works only in parallel Viterbi decoding. The main CPU has following functions: receive codeword, control the sub-CPU in synchronization, do Viterbi decoding parallel with the sub-CPU, send out decoding information, and determine whether decoding is finished or not. So the main CPU is the control CPU of the parallel Viterbi decoding system.

When the data transmission rate is higher than the decoding speed, codewords are temporarily stored in a RAM. After solving the previous group of codewords,



(a) Off-line simulation mode.

(b) On-line simulation mode.

Fig. 9 Simulation system for the parallel Viterbi decoding by multi-microprocessors.

CPUs process the present group of codewords stored in the RAM. When the transmission rate is lower than the processing speed, the CPUs will wait for the codeword.

5. 2 Results

As the Viterbi decoding has a fixed decoding speed, the channel error rate do not affect the decoding speed. The process speed of the parallel Viterbi decoding by multi-microprocessors is determined in following two ways:

1. Calculation of the programs execution time. Because programs are designed by assembly mnemonic, every instruction has known execution time. By computing the execution time of instructions in the decoding programs we can know the decoding

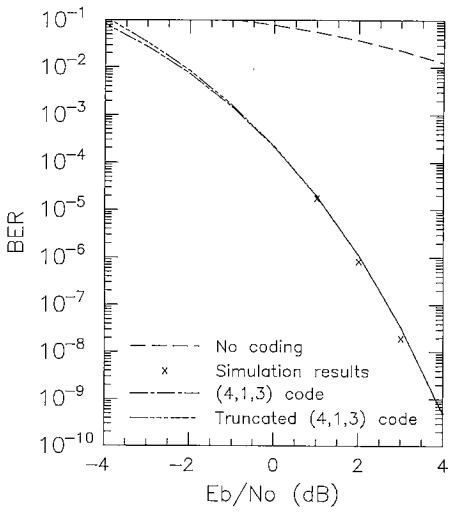


Fig. 10 Performance of the parallel Viterbi decoding.

speed.

2. Simulation test. The decoding speed is directly measured using a simulated input signal.

Simulation system is made by multi-microprocessors system as shown in Fig. 9, where the data generator, the encoder, the transmission channel and the decoder are simulated by microprocessors. The upper panel of Fig. 9 shows the off-line mode, with which the performance and speed can be analyzed. The lower panel of Fig. 9 shows the on-line mode, where the encoder and the decoder can be connected to other hardware for realtime applications. A software pseudo-noise generator was used to simulate digital noise on the binary symmetric channel (BSC). The channel error probability is chosen by the user during the measurements. In order to obtain good statistical measurements the number of samples measured should be very large. In this experiment, 1 Mbit samples are measured.

Figure 10 shows the decoder performance of the proposed scheme. The performance deterioration from the VA to the truncated VA is very small (less than 0.1 dB at $E_b/N_o=3$ dB) and the simulation results of the performance are very close to the theoretical calculation. From the calculation and simulation, the speed of the serial decoding of one microprocessor is 20 kbits/s. For parallel Viterbi decoding of two CPUs, the decoding speed can reach 38 kbits/s. It is expected from this experiment that for the parallel Viterbi decoding of 8 CPUs the decoding speed can reach 120 kbits/s.

6. Conclusion

In this paper the implementation of parallel Viterbi decoding and the problems which need to be considered in parallel Viterbi decoding by microprocessors have been discussed. Multiple microprocessor architecture is used which results in a decoder of moderate complexity and cost as compared to a specially designed Viterbi decoder using a VLSI. The presented method of implementing the Viterbi algorithm allows the use of hardware with a limited processing speed to achieve a high throughput rate.

Acknowledgment

The authers wish to thank Drs. Y. Hirata, Y. Yasuda of KDD, and Dr. M. Ohashi of KDD R & D Laboratories for their fruitful discussions and helpful suggestions.

References

(1) Viterbi, A. J., "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inf. Theory*, vol. IT-13, pp. 260–269, Apr. 1967.

(2) Forney, Jr., G. D., "The Viterbi algorithm," *Proc. IEEE*, vol. 61, pp. 268–278, Mar. 1973.

(3) Frenette, N. J. P., Mclane, P. J., Peppard, L. E. and Cotter, F., "Implementation of a Viterbi processor for a digital communications system with a time-dispersive channel," *IEEE J. Sel. Areas Commun.*, vol. SAC-4, pp. 160–167, Jan. 1986.

(4) Gulak, P. G. and Shwedyk, E., "VLSI structures for Viterbi receivers," *IEEE J. Sel. Areas Commun.*, vol. SAC-4, pp. 142–154, Jan. 1986.

(5) Fettweis, G. and Meyr, H., "Parallel Viterbi algorithm implementation: Breaking the ACS-bottleneck," *IEEE Trans. Commun.*, vol. COM-37, pp. 785–789, Aug. 1989.

(6) Gulak, P. G. and Kailath, T., "Locally connected VLSI architectures for the Viterbi algorithm," *IEEE J. Sel. Areas Commun.*, vol. SAC-6, pp. 527–537, Apr. 1988.

(7) Schweikert, R. and Vinck, A. J., "A convolutional single-parity-check concatenated coding scheme for high-data-rate application," *IEEE Trans. Commun.*, vol. COM-39, Jan. 1991.

(8) Conan, J., "An f8 microprocessor-based breadboard for the simulation of communication links using rate 1/2 convolutional codes and Viterbi decoding," *IEEE Trans. Commun.*, vol. 31, pp. 165–171, Feb. 1983.

(9) Said, S. M. and Dimond, K. R., "Realtime implementation of Viterbi decoding algorithm on a high-performance microprocessor," *Microprocessor and microsystem*, vol. 10, pp. 11–16, Jan./Feb. 1986.

(10) Kobayashi, N., Ohnishi, M., Koya, M., Tsukamoto, N. and Kokuryou, Y., "Simplified Viterbi decoding for high-speed data modem," *Trans. IEICE*, vol. J72-B-I, pp. 667–674, Aug. 1989.

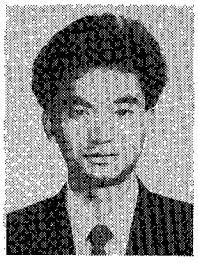
(11) Zhao, H., "Parallel Viterbi decoding implementation by use of microprocessors," *Masters dissertation, Shanghai University of Science and Technology*, Shanghai, P. R. China, Jun. 1988.

(12) Lin, S. and Costello, Jr., D. J., *Error Control Coding: Fundamentals and Applications*, Prentice-Hall, New Jersey, 1983.

(13) Omura, J. K., "On the Viterbi decoding algorithm," *IEEE Trans. Inf. Theory*, vol. IT-15, pp. 177–179, Jan. 1969.

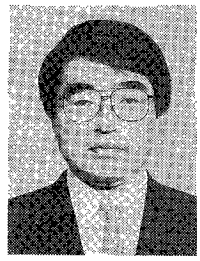
(14) Forney, Jr., G. D., "Maximum-likelihood sequence estimation of digital sequences in the presence of intersymbol interference," *IEEE Trans. Inf. Theory*, vol.IT-18, no. 3, pp. 363–378, May 1972.

(15) Viterbi, A. J. and Omura, J. K., *Principles of Digital Communication and Coding*, McGraw-Hill, New York, 1979.
(16) Perrott, R. H., *Parallel Programming*, Addison-Wesley Publishing Company. Inc., 1987.
(17) Horbest, E., Muller-Schloer, C. and Schwartzel, H., *Design of VLSI circuits*, Based on VENUS, Springer-Verlag, New York, 1987.



Hui Zhao was born on April 14, 1961. He received the B.S. degree in electrical engineering from Harbin Institute of Technology, Harbin, P. R. China in 1982 and the M.S. degree in electrical engineering from Shanghai University of Science and Technology, Shanghai, P. R. China in 1988. From 1988 to 1990, he was employed in the Shanghai Research Institute of Radio Equipment, the Ministry of Aeronautic and Astronautic Industry (MAAI), P. R. China, involved in the digital communication system devoloplent. He is now studying towards the Ph.D. degree at the Department of Electrical Engineering II, Faculty of Engineering, Kyoto University, Kyoto, Japan. His current research interest includes error control techniques, coding theory and satellite communication.

Xiaokang Yuan was born in Sept. 1939. He received the B.S. degree in the electric engineering from University of Electronics Science and Technology of China, Chengdu, P. R. China in 1960. Since 1960 he has joined the Ministry of the Aeronautic and Astronautic Industry (MAAI), P. R. China, as a research engineer involved in microwave system techniques and digital communication system developlent. Now he is a Professor and a head of the Department of Communication Engineering in the Shanghai Research Institute of Radio Equipment, MAAI, P. R. China. He is also a concurrent-Professor at Peikin University of Aeronautics and Astronautics, Peikin, P. R. China. His research interests include microwave system techniques, digital Communication system and spread spectrum system. He is a senior member of the Institute of Electronics, P. R. China.



Toru Sato was born on March 27, 1954. He received the B.E., M.E., and Ph. D. degrees in electrical engineering from Kyoto University, Kyoto, Japan in 1976, 1978, and 1982, respectively. He was a graduate student research associate of Arecibo Observatory, National Astronomy and Ionosphere Center from 1979 to 1980. He joined Radio Atmospheric Science Center of Kyoto University in 1983 as a research associate. He has been a lecturer at Department of Electrical Engineering II, Kyoto University since 1988. His major research interests have been system design and signal processing for atmospheric radars, radar remote sensing of the atmosphere, observations of precipitation using radar and satellite signals, radar observation of space debris, signal processing for subsurface radars, and digital satellite communication. He was awarded Tanakadate Prize in 1986. He is a member of the Society of Geomagnetism and Earth, Planetary and Space Sciences, the Japan Society for Aeronautical and Space Sciences, the Institute of Electrical and Electronics Engineers, and American Meteorological Society.

Iwane Kimura was born on December 25, 1932. He received the B.E., M.E., and Ph.D. degrees in electrical engineering from Kyoto University, Kyoto, Japan in 1955, 1957, and 1961 respectively. Since 1960, he has been a staff of Kyoto University; Department of Electronics and Department of Electrical Engineering II where he is now a Professor radio engineering since 1971. He has also been a visiting Professor of Institute of Space and Astronautical Science from 1981 to 1991. From 1964 to 1965, he was a research associate at Radioscience Laboratory, Stanford University on leave from Kyoto University. His research interests have been in radio science; particularly remote-sensing of upper propagation and generation of radio waves in magnetospheric and ionospheric plasmas, plasma measurements by rocket-Doppler technique, and active wave experiments in space plasmas using scientific satellites. He was awarded Inada Memorial Prize in 1958, and Tanakadate Prize in 1961. He is a member of the Institute of Electrical Engineers of Japan, the Society of Geomagnetism and Earth, Planetary and Space Sciences, and the American Geophysical Union.