Adapting the strongly-connected trellis concept for use with trellis-coded modulation

Sébastien Roy[1] and Paul Fortier[2]

[1]Systèmes Tertius Oculus
RR 2, Site 17, Box 9, Beresford, NB, E0B 1H0

[2]Département de génie électrique
Université Laval, Sainte-Foy, QC, G1K 7P4
fortier@gel.ulaval.ca

Abstract. The strongly-connected trellis was originally proposed by Chang and Yao as a way of increasing the efficiency of Viterbi decoders implemented on locally-connected parallel processor arrays. Originally, this concept was derived with binary convolutional codes in mind. Its adaptation to TCM is non-trivial due to the presence of parallel branches in the trellis. The method we propose utilizes a modified strongly-connected trellis concept to increase the efficiency of TCM decoders on locally-connected processor arrays. A TCM decoder based on these principles was implemented on a MasPar massively parallel computer for experimental purposes.

1 Introduction

The complexity of the Viterbi algorithm is proportional to the coding gain desired. In fact, obtaining high coding gains at high symbol rates requires enormous computational power. This has lead to increased interest in parallel implementations of the Viterbi algorithm, parallel processing being the last possible way to increase processing speed once the physical barrier has been reached in VLSI. Modern VLSI Viterbi decoders are made up of parallel processing elements and can provide high throughput for codes up to length $K = 7$.

Unfortunately, it would appear that the Viterbi algorithm is difficult to implement in parallel [1]. Indeed, the algorithm cannot be divided into independent sub-tasks and calls for high connectivity between processing elements. In VLSI, links occupy more space than transistors and high degrees of connectivity can be very costly. Indeed, VLSI design criterions call for regular, repetitive circuit patterns with local links only (systolic arrays). Physical limits are quickly reached since the number of links necessary for efficient Viterbi decoding grows exponentially with the constraint length. For this reason, it is interesting to consider locally-connected methods even if it implies a loss of efficiency.

Chang and Yao [2] have proposed the strongly-connected trellis in 1989 as a way to increase the efficiency of Viterbi decoding on locally-connected processor arrays. This concept was originally intended for binary convolutional codes. We propose a method to adapt the strongly-connected trellis to the more powerful trellis-coded modulation schemes [3, 4]. Using these principles, a software decoder for TCM has been constructed on a MasPar MP1 massively parallel computer.

While the application of the strongly-connected trellis to parallel Viterbi decoding leads to increased efficiency for binary convolutional codes, it is shown herein that similar benefits can be derived for parallel decoding of trellis-coded modulation schemes.

This paper is organized as follows. Section 2 discusses the architecture of the MasPar computer. In Section 3, the Viterbi algorithm is presented as a matrix operation. Section 4 introduces the concept of the strongly-connected trellis, while its application to TCM is presented in Section 5. In Section 6, an implementation of the parallel Viterbi algorithm to decode TCM on the MasPar is shown. Finally, the paper concludes in Section 7.

2 Parallel architectures and the Maspar computer

Many types of parallel architectures exist and each is programmed differently. The MasPar MP1 [5, 6] is made up of 2048 processor elements (PEs) arranged in a rectangular matrix of 64 X 32. Each PE is connected to its eight nearest neighbors by high-speed links (see figure 1). Furthermore, processors on the edges of the array are linked to the opposite edge by wraparound links. The processor array is in fact toroidal in shape if we consider the presence of the wraparound links. These local connections provide a bandwidth of 24 Gigabytes per second. In addition, a global router mechanism is provided to permit random point-to-point linking. Such global links are much slower at 1.5 Gigabyte per second and important setup delays are incurred whenever a new global link is established. Since our work is concerned with locally-connected processor arrays, we have chosen not to use the global router.

The MasPar is a SIMD machine (Single Instruction Multiple Data). This means that the active portion of the processor array executes the same instructions at the same time. However, each processor operates on its own local dataset. This type of parallel architecture is not as powerful or as flexible as the MIMD class (Multiple Instruction Multiple Data) where each PE is an independent computer with its own memory and program (see figure 2). On the other hand, SIMD machines are cheaper and much easier to program since the software retains a sequential flow.

With any parallel computer, maximum efficiency is almost impossible to achieve because parallelism itself and interprocessor communications constitute an additional overhead. In fact, the efficiency tends to decrease as the number of processors increases [7]. Certain problems are better suited to certain architectures. It is important to point out that the SIMD architecture of the MasPar is very specialized and is best suited to problems involving a large number of independent sub-tasks such as image processing.

Let us take a look at Fast Fourier Transforms as a problem for SIMD computers. The FFT algorithm is characterized by the same type of interdependence pattern as the Viterbi algorithm. Because of this, it is not a good algorithm for parallelisation on any architecture, but especially SIMD. While running a single FFT problem on the MasPar's 2048 processors would be much faster than on most sequential machines, it would probably not be very efficient. However, if you happen to have a large number of such problems to resolve, it is much more efficient to run 2048 FFT's simultaneously. This approach is better suited to SIMD since each processor runs its own independent yet identical problem.

Since our goal is to accelerate a single TCM decoder and not 2048 such decoders, our problem is a poor candidate for SIMD parallelisation. The strongly-connected trellis concept helps to attain the maximum efficiency possible on a locally-connected architecture (SIMD or MIMD).

Our parallel implementation is unidimensional; in other words, all states in a single stage are treated simultaneously but each stage is treated sequentially. In this manner, we are making a better use of the number of processors and the architecture of the MasPar computer [8].

3 Parallel Viterbi decoding using matrix operations

Let us define an adjacency matrix $\mathbf{A}_k$ that represents the trellis stage corresponding to the *k*th transition. The element $a_k[i, j]$ holds the metric of the branch form state *i* to state *j*. Elements for which no branch exists in the trellis are given an infinite (or very large) value (see figure 3).

The add-compare-select step of the Viterbi algorithm can be redefined with the help of a matrix-vector operation similar in structure to matrix-vector multiplication. This approach has been helpful in the process of parallelizing the Viterbi algorithm since matrix-vector multiplication is well adapted to parallel architectures and extensively covered in the literature [7].

Given a vector $\vec{P}_k$ of *N* elements containing the lengths of the survivors after the *k*th transition, the matrix-vector form of the survivor update operation can be written as:

$$\vec{P}_{k+1} = \mathbf{A}_{k+1} \otimes \vec{P}_k$$

where the ⊗ operator is defined as follows:

$$p_{k+1}[i] = \min(a_{k+1}[i,j] + p_k[j]) \qquad j = 0, 1, \ldots, N-1$$

Let us compare equation (2) with the definition of matrix-vector multiplication:

$$p_{k+1}[i] = \sum_j (a_{k+1}[i,j] \times p_k[j]) \qquad j = 0, 1, \ldots, N-1$$

It can be verified that the ⊗ operator is structurally equivalent to matrix-vector multiplication. The sum and multiplication operations in the definition of matrix-vector multiplication have merely been replaced by the minimum operator and addition, respectively.

We need also know from which state the new survivors come so we must compute for each survivor *i*:

$$\hat{j}[i] = \min^{-1}(a_{k+1}[i,j] + p_k[j]) \qquad j = 0, 1, \ldots, N-1$$

Now that the core of the Viterbi algorithm has been formally defined as a matrix-vector operation, we need to define parallel algorithms based on equation (2). Given a linear array of *N* processors with wraparound (ring array), the vector-matrix operation can be executed in o(*N*) steps for an *N*-state convolutional code. Each processor initially contains an element of vector $\vec{P}_k$. Elements of matrix $\mathbf{A}_{k+1}$ are entered form the top as illustrated in figure 4. Each processor adds its element of $\vec{P}_k$ to its element of $\mathbf{A}_{k+1}$ and stores the result in a local accumulator. As the next set of values of $\mathbf{A}_{k+1}$ are entered, the values of $\vec{P}_k$ are rotated one processor to the right. Those values are again summed individually by each processor and the result is compared with the content of the local accumulator. The smallest value is stored in the accumulator. After N steps, the accumulators contain the elements of vector $\vec{P}_{k+1}$.

While the method described above is simple and elegant, it is desirable to exploit the power of more than *N* processors. A second method, called the division and fusion algorithm, can be scaled to fit processor arrays of various sizes. The available processor array is initially divide in *M* groups of *N* processors (see figure 5). Each group performs only a fraction of the matrix-vector operation. The first group deals exclusively with the first $\frac{N}{M}$ steps of the matrix-vector operation and so needs only generate the corresponding fraction of the branch metrics. The final result is merely the minimum of the *M* partial results for each element of $\vec{P}_{k+1}$.

4 The strongly-connected trellis

The performance of Viterbi decoders implemented on systolic arrays or similar parallel local connectivity structures suffers from inherent inefficiency. This is due to the fact that most convolutional codes result in a trellis with low connectivity. This in turn leads to sparse-matrix operations by the parallel Viterbi decoder and, therefore, inefficient use of processing resources.

To overcome this problem, Chang and Yao [2] have proposed to combine a number *r* of stages into one to obtain a strongly-connected trellis where all state-pairs are linked by one branch. There is a one-to-one correspondence between each branch of the strongly-connected trellis and each *r*-branch path in the primitive trellis (see figure 6). It has also been shown that the generation of metrics for these composite branches was straightforward and involved little additional overhead. The composite metric is merely the sequential concatenation of the individual branchs' metrics.

Let us define the compression ratio r_{max} as the maximum number of stages of the primitive trellis that can be compressed into one without creating ambiguity, i.e. losing the one-to-one correspondence between the primitive trellis and the strongly-connected trellis:

$$r_{max} = \mathrm{L}(\log_b N)$$

where *b* stands for the number of branches entering or leaving any state (a measure of the connectivity of the primitive trellis) and *N* stands for the number of state. The L() function rounds downward to the nearest integer. For codes with a bit rate of $\frac{1}{n}$, *b* is always equal to 2 since each state transition results of a single bit entering the encoder. Such codes have low-connectivity trellises which implies high compression ratios. For example, a 256 states rate $\frac{1}{n}$ convolutional code will allow a compression of eight stages, resulting in a fully-connected trellis.

On the other hand, TCM codes are usually built around a rate $\frac{n}{n+1}$ in which case *b* will be equal to n^2 in equation (5). Indeed, TCM is characterized by high trellis connectivity. Accordingly, the compression ratio r_{max} will typically be smaller for TCM codes than binary convolutional codes with the same number of states *N*.

5 Trellis-coded modulation

Consider a 256-states 16-QASK TCM encoder as depicted in figure 7. The signal set is partitioned in eight subsets, each of which contains two maximally distant symbols [3, 4]. The output of a rate $\frac{2}{3}$ convolutional encoder is used to select one of the eight subsets while a single message bit selects the symbol to be transmitted within the subset. The corresponding trellis is characterized by pairs of parallel branches while the overall connectivity remains low.

The standard decoding procedure for TCM comprises three steps:

1- Metrics are generated for all branches in a trellis stage based on unquantized channel output and Euclidean distance (soft decision).

2- The metrics for each group of parallel branches are compared and all but the shortest branches for each state-pair are eliminated.

3- The trellis being reduced to single branches, conventional Viterbi decoding can now be applied.

Incorporating the strongly-connected trellis concept in the TCM decoding process is non-trivial due to the presence of parallel branches in the original trellis. This obstacle can be avoided by eliminating parallel branches (steps 1 and 2 described above) at the primitive trellis level. Therefore, a reduced trellis, ridden of its parallel branches can be “compressed” instead of the original one (see figure 8). The reduced trellis is actually a representation of the subset part of the encoder where no information remains about which signal was selected within the subset.

From this point, decoding can proceed efficiently using the Viterbi algorithm at the strongly-connected trellis level.

6 Implementation

Initially, a table must be constructed where for each branch of the strongly-connected trellis, the corresponding path or transition sequence in the original trellis can be looked up. Since the reduced trellis is considered at this point, the transition sequence is identified by its subsets and is thus unique. The table can take the form of an *N* X *N* matrix with column and row indices corresponding to starting and ending states while the value itself reveals the in-between transition sequence. While such a data structure is rather large, it should be distributed to a number of local processor memories. In fact, each processor will only need to access its own local portion of the branch-to-path table. The purpose of this table is to provide a link between the primitive trellis and the strongly-connected trellis, allowing work to be conducted at both levels.

The decoding algorithm can be mapped to a parallel processing array in a number of ways. We have used a partitioning scheme based on ending states, i.e. processor 0 is assigned all branches which end at state 0. Local memories need only contain the portions of the branch-to-path table corresponding to their assigned branches.

For the TCM code described above, there are 4 branches entering or leaving any state in the reduced trellis. This results in a compression ratio of $r_{max} = \log_4 256 = 4$. Consequently, the decoding is conducted on groups of 4 symbols. For each composite branch, the corresponding subset sequence is looked up in the branch-to-path table and eight branch metrics are then computed since there are 4 transitions and 2 parallel branches per transition. The smallest metrics for all 4 transitions are added up to form the composite branch metric.

During the reduction phase, note must be taken of which branch was selected within each group of parallel branches. For a given strongly-connected trellis branch, a transitional code gives this information for each branch of the corresponding path in the primitive trellis. In our example, the transitional code would consist of a 4-bit sequence in memory where each bit identifies the branch retained for each of the four transitions. This transitional code is illustrated in figure 9. Once the metrics and transitional codes for all branches of the strongly-connected trellis have been generated, Viterbi decoding can be performed using matrix operations.

When the end of a data frame is reached, the Viterbi decoder yields a state sequence. The survivor memory containing this state sequence also contains an associated transitional code sequence. The state sequence provides information at the subset level while the transitional code sequence identifies the signal within each subset. These two sequences are combined to obtain a maximum-likelihood signal sequence from which the original message can be derived.

Figure 10 gives a general block diagram of the parallel TCM decoding algorithm based on the strongly-connected trellis concept. One may observe that metric generation is performed at the primitive trellis level while Viterbi decoding itself (the add-compare-select step, the bulk of the work) occurs at the strongly-connected trellis level.

A simulation was conducted on the MasPar computer using a 16-QASK 256 states TCM code. Using all 2048 processors, a decoding rate of 190 message bits per second was obtained. We should point out that this is the performance of a software decoder, written in a high level language, and it is therefore much slower than comparable custom VLSI decoders. Previously, a similar simulation based on the strongly-connected trellis had been performed using a standard 256-states binary convolutional code - a decoding rate of 1320 bits per second was measured [9]. This performance is 69.5 times better than a sequential decoder running on a Sun SPARCstation IPC (19 bits per second).

On the other hand, the TCM simulation on the MasPar was 6.95 times slower than the binary convolutional code simulation with the same number of states. This slowdown is imputable to the higher computational complexity of the TCM code. Specifically:

- Floating point arithmetics are used for metrics instead of integer arithmetics.
- The Euclidean distance criterion is used instead of Hamming distance
- Mapping message bits to modulated signals and vice-versa constitutes an additional overhead.
- The metric generation step has to be performed at the primitive trellis level.

In theory, for a binary convolutional code with 256 states and rate $\frac{1}{2}$, the use of the strongly-connected trellis speeds execution by a factor of 8 ($\log_2 256$). On the other hand, the theoretical speedup factor for a 16-QASK TCM code with 256 states is only 4. This is an additional justification for the performance gap between the two types of codes.

7 Conclusion

In this paper, we have shown that the strongly-connected trellis concept can be successfully applied to TCM codes and yield efficiency gains comparable to those obtained with binary convolutional codes. Even though the actual implementation was done on a particular computer, namely the MasPar MP1, its systolic-like architecture serves as a good model for possible VLSI implementation.

References

[1] J. Sparso, S. Pederson and E. Paaske, “Design of a Fully Parallel Viterbi Decoder,” Proc. VLSI 91, pp. 2.2.1-2.2.10, 1991.

[2] C. Y. Chang and K. Yao, “Systolic Array Processing of the Viterbi Algorithm,” IEEE Trans. Inform. Theory, vol. 35, pp. 76-86, 1989.

[3] G. Ungerboeck, “Channel coding with multilevel/phase signals,” IEEE Trans. Inform. Theory, vol. 28, pp. 55-67, 1982.

[4] G. Ungerboeck, “Trellis-coded modulation with redundant signal sets - Part I: Introduction,” IEEE Communications Magazine, vol. 25, no. 2, pp. 5-11, 1987.

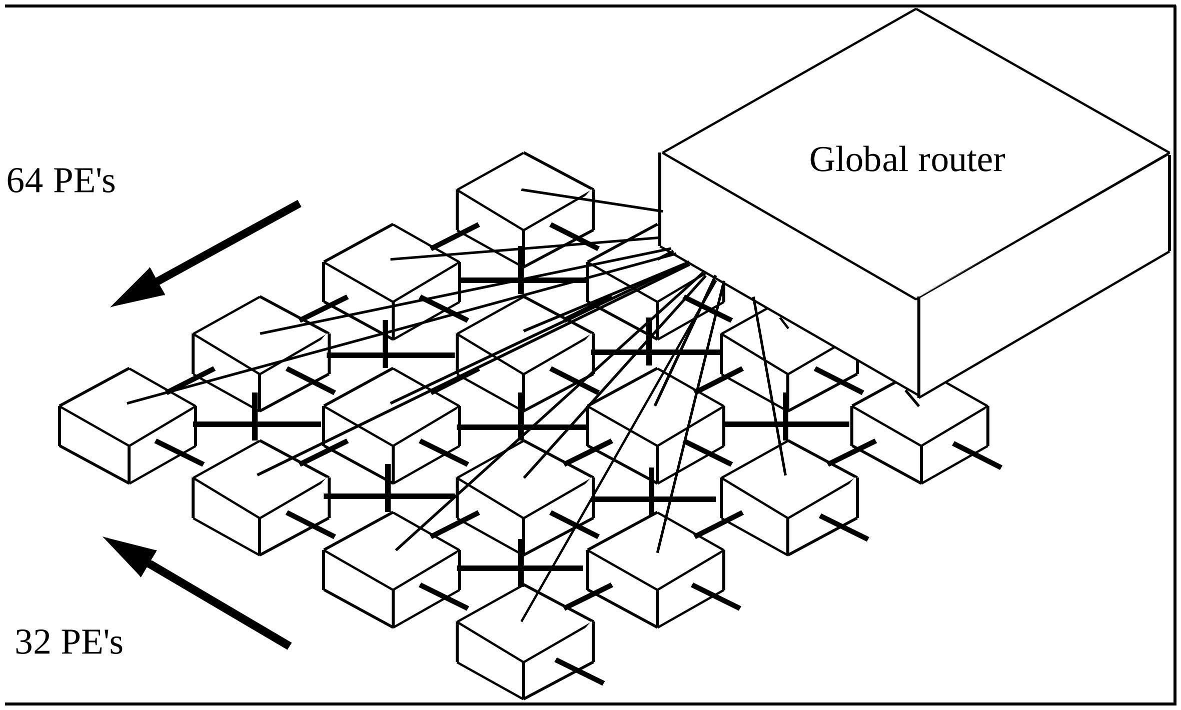
[5] --, MasPar Parallel Application Language (MPL) User Guide. Sunnyvale: MasPar Computer Corporation, 1991.

[6] --, MasPar Parallel Application Language (MPL) Reference Manual. Sunnyvale: MasPar Computer Corporation, 1991.

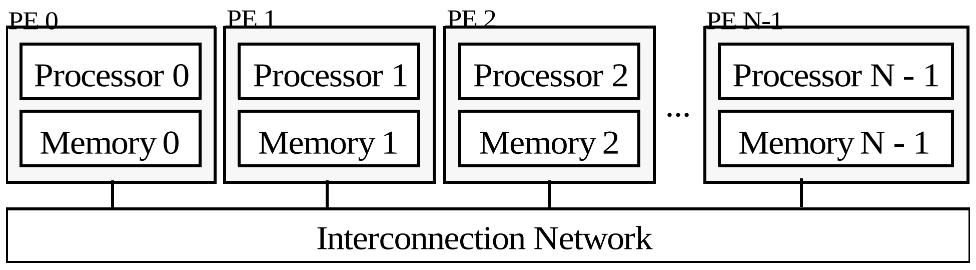
[7] F. T. Leighton, Introduction to Parallel Algorithms and Architectures. San Mateo, CA: Morgan-Kaufmann, 1992.

[8] S. Roy, Méthodes de réalisation de l’algorithme de Viterbi sur un ordinateur massivement parallèle de type SIMD, Masters’ thesis, Univ. Laval, Sainte-Foy, QC, 1993.

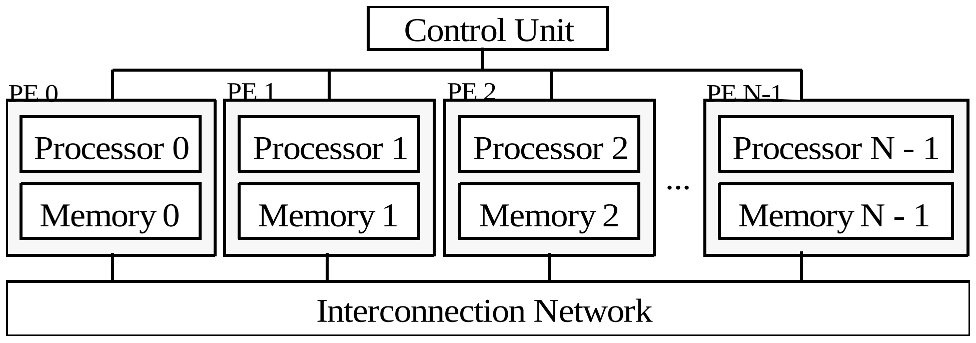
[9] S. Roy and P. Fortier, “An implementation of the Viterbi algorithm on a massively parallel computer”, Proc. CCECE ‘92, Toronto, Canada, pp. MM10.2.1-MM10.2.4, 1992.

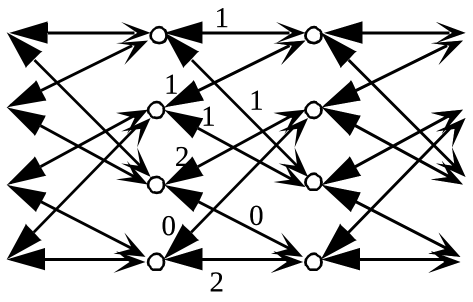

Global router
64 PE's
32 PE's

1. MIMD


PE 0
PE 1
PE 2
PE N-1
Processor 0
Processor 1
Processor 2
Processor N - 1
Memory 0
Memory 1
Memory 2
Memory N - 1
...
Interconnection Network

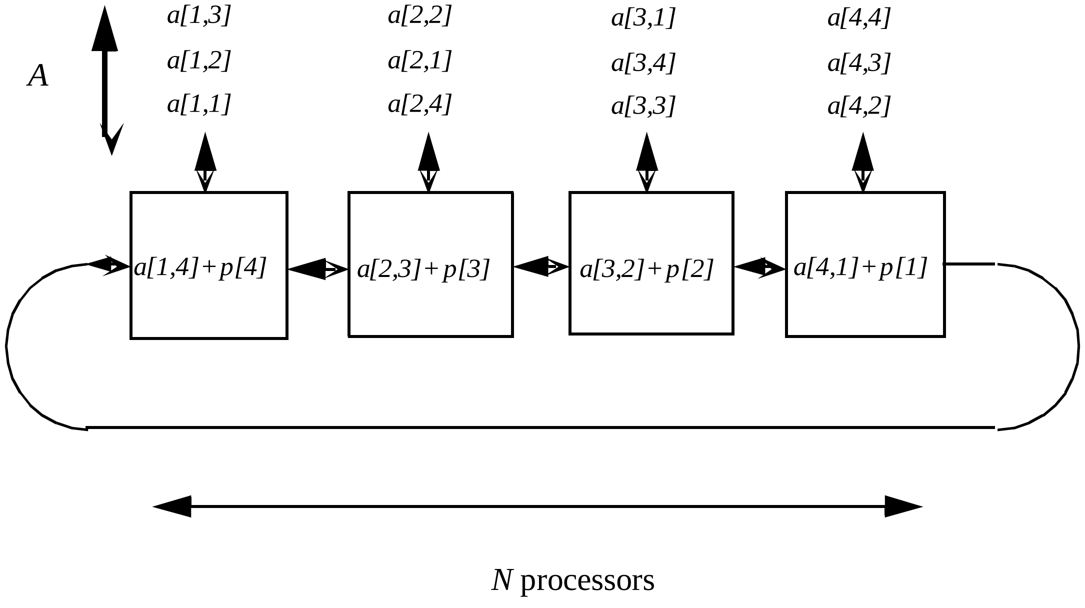
2. SIMD

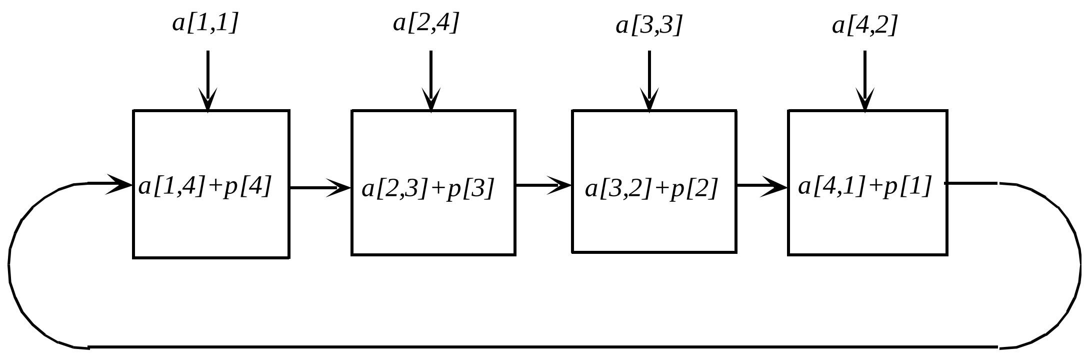

Control Unit
PE 0
PE 1
PE 2
PE N-1
Processor 0
Processor 1
Processor 2
Processor N - 1
Memory 0
Memory 1
Memory 2
Memory N - 1
...
Interconnection Network

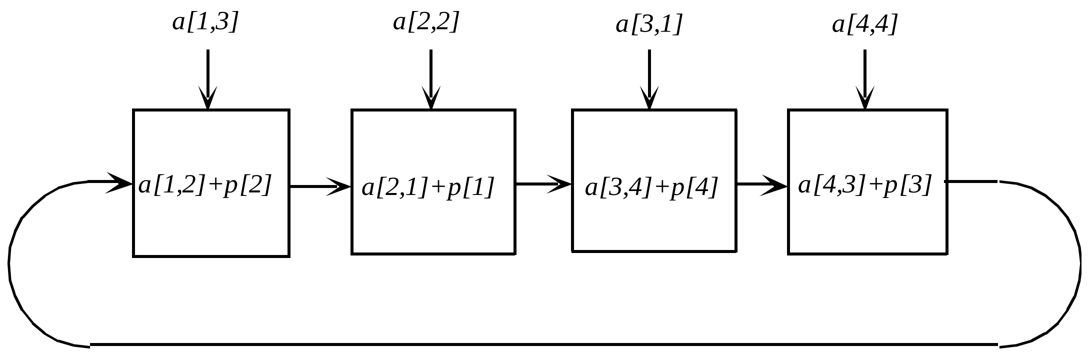


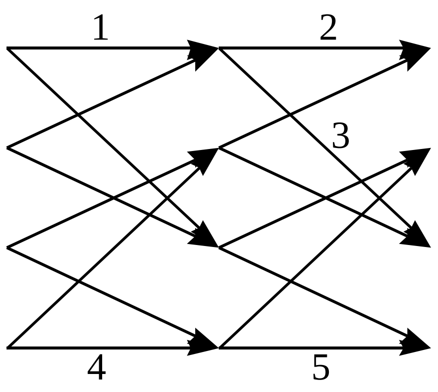
k k+1

$$A_{k,\,k+1} = \begin{bmatrix} 1 & \infty & 1 & \infty \\ 1 & \infty & 1 & \infty \\ \infty & 0 & \infty & 2 \\ \infty & 0 & \infty & 2 \end{bmatrix}$$

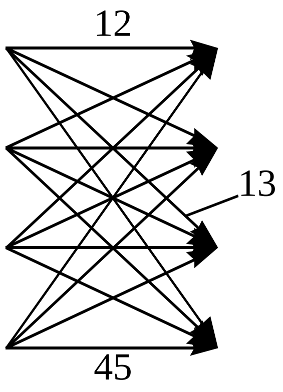

A
a[1,3]
a[1,2]
a[1,1]
a[2,2]
a[2,1]
a[2,4]
a[3,1]
a[3,4]
a[3,3]
a[4,4]
a[4,3]
a[4,2]
a[1,4]+p[4]
a[2,3]+p[3]
a[3,2]+p[2]
a[4,1]+p[1]
N processors


a[1,1]
a[2,4]
a[3,3]
a[4,2]
a[1,4]+p[4]
a[2,3]+p[3]
a[3,2]+p[2]
a[4,1]+p[1]

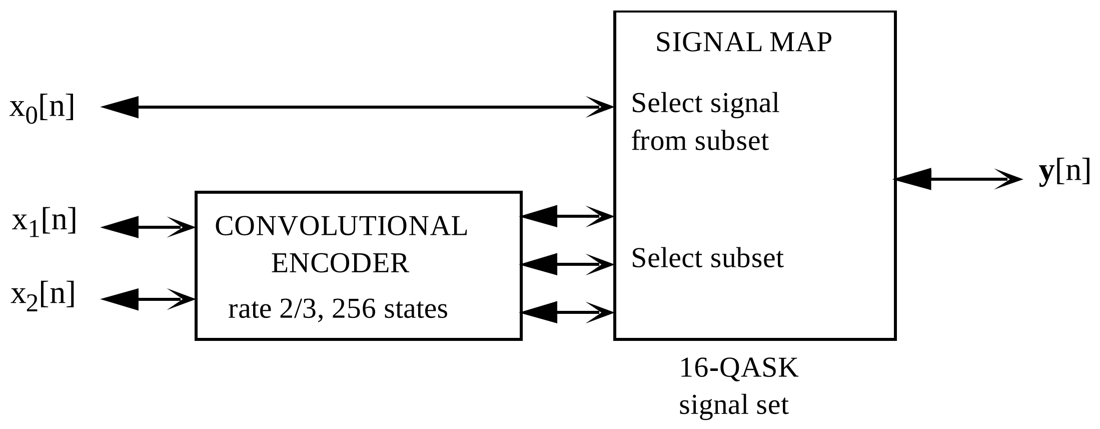

a[1,3]
a[2,2]
a[3,1]
a[4,4]
a[1,2]+p[2]
a[2,1]+p[1]
a[3,4]+p[4]
a[4,3]+p[3]

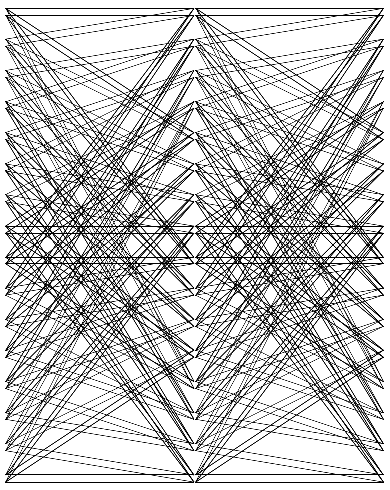


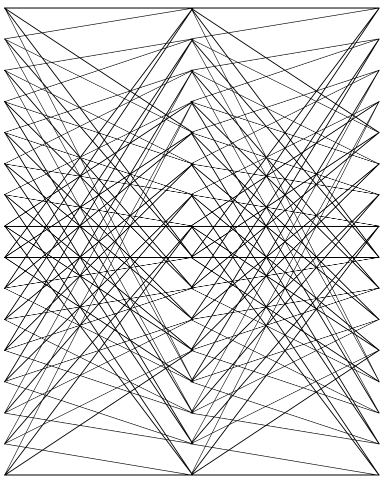
1
2
3
4
5

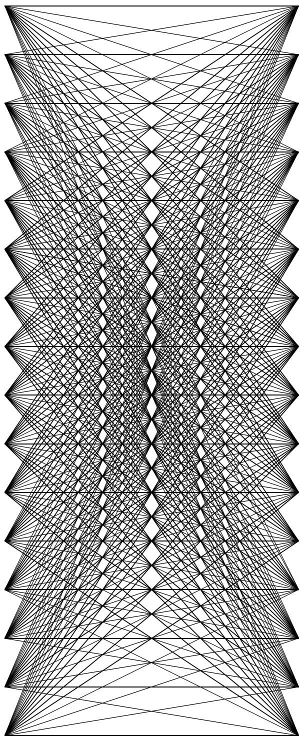


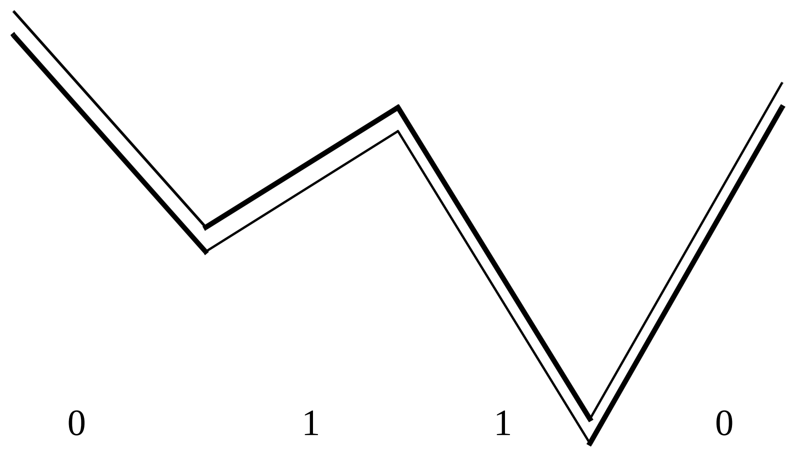
12
13
45

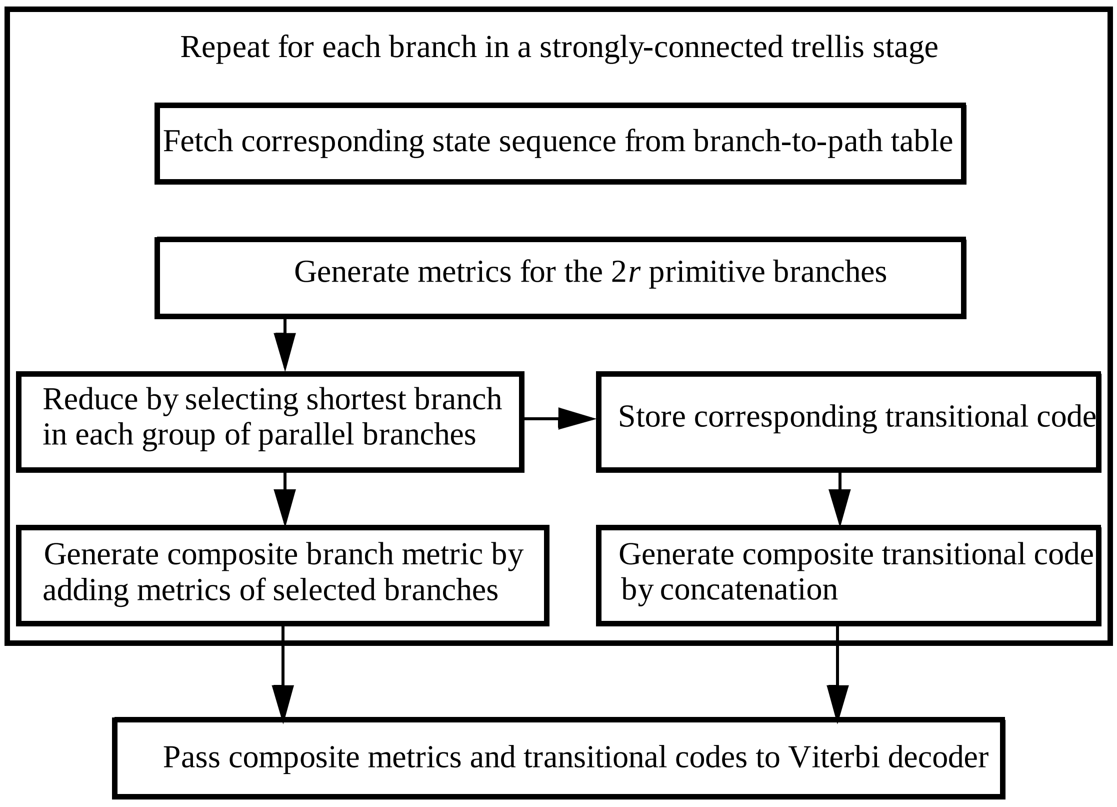

SIGNAL MAP
Select signal from subset
Select subset
$x_0[n]$
$x_1[n]$
$x_2[n]$
CONVOLUTIONAL ENCODER
rate 2/3, 256 states
$\mathbf{y}[n]$
16-QASK signal set


(a) primitive


(b) reduced


(c) compressed


0
1
1
0


Repeat for each branch in a strongly-connected trellis stage
Fetch corresponding state sequence from branch-to-path table
Generate metrics for the 2*r* primitive branches
Reduce by selecting shortest branch in each group of parallel branches
Store corresponding transitional code
Generate composite branch metric by adding metrics of selected branches
Generate composite transitional code by concatenation
Pass composite metrics and transitional codes to Viterbi decoder