Chapter 2

MLSDA Algorithms and Systolic Priority Queue Algorithms

In this chapter we introduce the decoding algorithm MLSDA, systolic systems briefly. The priority queue structures that will be modified to implement MLSDA are also presented.

2. 1 Maximum-Likelihood Sequential Decoding Algorithm

Let $\boldsymbol{v} = (v_0, v_1, ..., v_{N-1})$ represent a binary codeword of an (n,k,m) convolutional code, where, m is the memory order, $N = n(L+m)$ is the code length and L is the length of the information sequence. Denote a portion of codeword $\boldsymbol{v}$ by

$$v_{(a,b)} = (v_a, v_{a+1}, ..., v_b)$$

Assume that the binary codeword is antipodally transmitted over a additive white Gaussian noise (AWGN) channel, and induce a received vector $r=(r_0, r_1, \ldots, r_{N-1})$. In other words,

$$r_j = (-1)^{vj}\sqrt{\varepsilon} + \lambda_j$$

where ε is the signal energy per channel bit, and λ_j is a noise sample of Gaussian process with single-sided noise power per hertz N_0. The variance of λ_j is $N_0/2$, and the signal-to-noise (SNR) for the channel is $\gamma = \varepsilon / N_0$. in order to account for the code redundancy for different code rates, we will use the signal-to-noise ratio per

information bit (SNR_b), denoted by γ_b, as

$$\gamma_b = \frac{\mathcal{E}_b}{N_o} = \frac{N}{KL}\gamma = \frac{1}{\widetilde{R}}\gamma$$

Represent an estimate of the transmitted codeword $\boldsymbol{v}$ by $\underline{v}$.

The maximum-likelihood decoding rule (MLD rule) [Ref. 3, 4, 5] for a time-discrete memoryless channel can be formulated as

$$\text{estimate } \underline{v} = v^* \quad \text{for} \quad v^* \in C,$$

if

$$\sum_{j=0}^{N-1}\left(\phi_j - (-1)^{v_j^*}\right)^2 \le \sum_{j=0}^{n-1}\left(\phi_j - (-1)^{v_j}\right)^2 \qquad \text{for all } v \in C \quad (2.2.1)$$

where

$$\phi_j = \ln\frac{\Pr(r_j \mid 0)}{\Pr(r_j \mid 1)}$$

In the special case where the codewords are transmitted with equal probability, the MLD rule minimizes the error probability. We refer the above MLD rule as *the first form of the MLD rule*.

Let $\boldsymbol{y} = (y_0, y_1, \ldots, y_{N-1})$ be the hard-decision of $\phi = (\phi_0, \phi_1, \ldots, \phi_{N-1})$, i.e., if $\phi_j < 0$ then $y_i = 1$, else $y_i = 0$.

Furthermore, let **s** be the syndrome of y, and H is a parity-check matrix for $\boldsymbol{C}$. That is,

$$s = yH^T$$

Denote the collection of all error patterns whose syndrome is **s** by E(**s**). Then another form of MLD rule can be formulated as follows which we call the second form of the MLD rule [Ref. 19]:

$$\text{estimate } \underline{v} = y \oplus e^* \quad \text{for} \quad e^* \in E(\mathbf{s}),$$

if

$$\sum_{j=0}^{N-1} e_j^* |\phi_j| \le \sum_{j=0}^{N-1} e_j |\phi_j| \qquad \text{for all } e \in E(\mathbf{s})$$

We next define a new path metric based on the second form of the MLD rule.

Definition 2.2.1:

For any path $\mathbf{x}_{(\ln-1)}$ ending at level l in a trellis, we define the metric associated with it as

$$M\left(x_{(ln-1)}\right)=\sum_{j=0}^{ln-1}\left(yj \oplus xj\right)\left|\phi j\right| \tag{2.2.2}$$

The term $(y_j \oplus x_j)|\phi_j|$ in (2.2.2) is called *bit metric*, and is denoted by $M(x_j)$.

Following the definition of the path metric, we define the metric of a code path ν by

$$M\left(v\right)=\sum_{j=0}^{ln-1}\left(yj \oplus xj\right)\left|\phi_j\right|$$

It is clear that $\mathbf{e} = y \oplus \nu \in E(\mathbf{s})$. Thus, to find an $\mathbf{e}^*$ in the second form of the MLD rule is equivalent to finding a code path with the smallest metric in the trellis. We herein have a sequential decoding algorithm using the new metric. As a result, the sequential decoding algorithm based on the new metric becomes an ML decoder. We therefore call it maximum-likelihood sequential decoding algorithm (**MLSDA**).

Unlike the conventional sequential decoding algorithm which requires only one stack, the **MLSDA** operating on a trellis needs to maintain two stacks, which are respectively named the Open Stack and the Close Stack. The Open Stack contains all paths ending at the frontier part of the trellis, which have been explored by the algorithm. The Closed Stack stores the information of the ending states and ending levels of the paths, which had ever been the top paths of the Open Stack. The **MLSDA** operating on a trellis is stated below.

<The trellis-based MLSDA>

Step 1. Load the Open Stack with the origin node whose metric is assigned zero.

Step 2. Compute the metrics of the successors of the top path in the Open Stack, and put the ending state and the ending level of the top path into the Closed

Stack. Delete the top path from the Open Stack.

Step 3. Whenever any of the new paths (i.e., the successors of the top path in the Open Stack in Step 2) merges with a path already in the Open Stack, eliminate the one with higher metric value. If any of the new paths merges with a path already in the Closed Stack, discard it.

Step 4. Insert the new paths into the Open Stack, and reorder the Open Stack according to ascending metric values.

Step 5. If the top path in the Open Stack ends at the terminal node in the trellis, the algorithm stops; otherwise go to Step 2.

The **MLSDA** operating on a trellis actually generates a search graph containing all paths in the Open Stack. Note that in Step 2, the ending state and the ending level of the top path is put into the Closed Stack so that we can check the merging of any two paths by using these information. It can be shown that the **MLSDA** is an ML decoding algorithm. The main issue of designing VLSI architecture of MLSDA is to perform efficiently the sorting associated to OPEN stack. The technology used in the thesis is the systolic priority queue method that will be described in the following sections and the next chapter.

2.2 Systolic algorithm and system

High performance computer system must rely on parallelism to achieve very high throughput. The systolic system concept, popularized by Kung [Ref. 6], allows effective use of a very large number of simple processors to operate in parallel. The processing elements, which have simple and regular interconnection, pass data between them in a regular, rhythmic manner with little or no global data communication. VLSI technology has made it feasible to layout an entire systolic

system on a single electronic chip or circuit board. Thus high performance systolic system can be implemented directly on low-cost hardware. After its invention a few years ago, the systolic system concept has gained tremendous popularity. Many special purpose systolic arrays have been proposed for solving problems in signal and image process, computational linear algebra, graph theory, data structure, pattern and language recognition, etc [Ref. 7]. Actual implementations of some of these arrays are already underway. Systolic systems are generalizations of iterative arrays [Ref. 8, 9, 10, 11] and cellular automata [Ref. 12, 13, 14], in which the processors have fixed memory (i.e., finite-state machines) rather than processors with unbounded memory. These arrays were used primarily as models for pattern and language recognition.

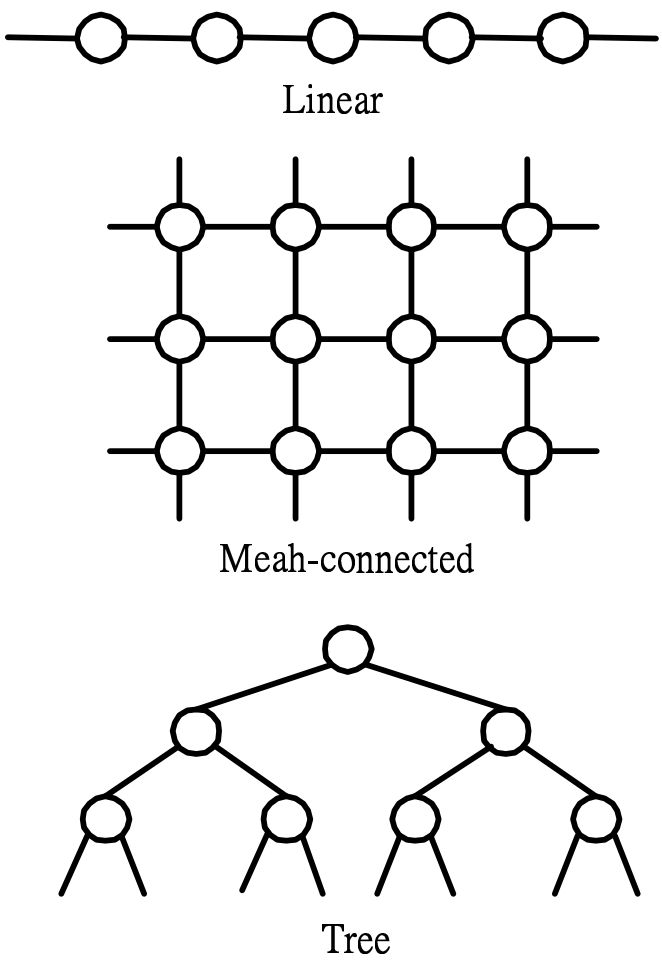

Fig. 2.1 Examples of systolic system

Some examples of the shapes and interconnections of systolic systems are shown in Fig. 2.1. There are several types of input/output modes. For example, in a mesh-connected array, the inputs can be pipelined to the left boundary cells and the outputs taken from the right boundary cells. Or the inputs can be fed serially to the lower left-hand corner and the serial outputs taken from the upper right-hand corner.

Several researchers have explored hardware implementation of data structures. Leiserson implemented a priority queue [Ref. 15] and Guibs and Liang have given linear systolic array implementation of queues and stacks [Ref. 16]. Atallah, Kosaraju [Ref. 17] and Somani, Agarwal [Ref. 18] have shown that dictionary operations can be implemented on a tree. In this thesis, we refer to the systolic structures to design a priority queue for sequential decoding.

The priority queue is the most important one in the proposed decoder because it decides the decoding speed. Systolic priority queue has the advantage of fast and constant searching speed. For this reason, we give up the traditional stack memory management technique, such as RAM, and use the systolic system for searching the best metric from a large set of metrics.

2. 3 The Systolic Priority Queue Algorithm

To explain the algorithm that rules node movement, we refer to the conceptual representation shown in Fig . 1.1 for codes with code rate R=1/2.

The processors are resembled as a truss bridge with trusses arranged in a triangular fashion. Each processor A_i, i=0,1,2…., can handle one node, except A_0, which acts as an input port, and A_1, which acts as an input and output port. As shown in the figure, each processor A_i, i=2,3,4,…, connects to its four neighbors, A_{i-2}, A_{i-1}, A_{i+1}, A_{i+2}. There

are three actions performing in the algorithm: shift right, shift left, and node exchange. Furthermore, let $\mathbf{A}_i$ be the processor that stores the node, $\mathbf{S}_i$ be the insert node and **T** be the extracted node. Hence, the algorithm can be described as follows:

<PESPQ algorithm>

Step 1. Insert: deliver S_0 to A_0 and S_1 to A_1.

Step 2. Shift right: send node from A_{i-1} to A_{i+1} i=1,2…(i.e. A_0 to A_2, A_1 to A_3, …).

Step 3. Sort triple: make A_{3i-1}, A_{3i} and A_{3i+1} to be a unit, i=1,2,3…. Send the node with best metric in the unit to A_{3i-1}.

Step 4: Shift left and extract: send A_{i+1} to A_i, i=1,2,…, and deliver T from A_1.

Step 5: Sort triple: the same as Step 3. Make A_{3i-1}, A_{3i} and A_{3i+1} to be a unit, i=1,2,3…. Send the node with best metric in the unit to A_{3i-1}.

As stated in the above algorithm, the queue accepts two new nodes in parallel and delivers the node with the best metric. The fact that the algorithm always delivers the top node with the best metric is simply a consequence of the following Theorem.

Definition 2.3.1:

Every node in Ai has a rank relative to the top node. The top node T has rank $\mathbf{r}_T$=1, the second largest node (metricwise) has rank $\mathbf{r}_T$=2, and so on. When several nodes have the same metric value, the smaller rank is assigned to the previous node in the queue. The rank of the node in the queue will be refreshed in every cycle.

Theorem 2.3.1:

After any cycle of the parallel entry systolic priority queue, the node with rank ***r*** , ***r***=1,2,.., lies in a processor $\mathbf{A}_k$ such that $2 \leqq k \leqq 3r-1$.

Proof [Ref. 1]**:** The theorem is proven by induction. Assuming that after N cycles $k \le 3r\text{-}1$ is correct, we show that after N+1 cycles this relation still holds. The ordering algorithm is examined step by step:

(1) $A_0 \leftarrow S_0$, $A_1 \leftarrow S_1$: inserting two new nodes cannot decrease the ranks of the nodes that are already in the queue. Since their positions remain unchanged, $k \le 3r\text{-}1$ remains correct.

(2) $A_{i+1} \leftarrow A_{i-1}$: after the shift, $k \le 3r\text{-}1$ is changed to $k \le 3r+1$.

(3) Sort triplets: if before the ordering a node N is in a processor A_k such that $k = 3_{rN}$ or $k = 3_{rN+1}$, then the triplet of processors A_{3i-1}, A_{3i} and A_{3i+1}, moves it to the position $k = 3_{rN-1}$. indeed the nodes in the two other processors must have larger ranks otherwise they would violate $k \le 3r+1$. After the ordering, $k \le 3r\text{-}1$ is valid again.

(4) $A_i \leftarrow A_{i+1}$: after the shift, $k \le 3r\text{-}1$ becomes to $k \le 3r\text{-}2$. The node in processor A_1 has rank one and is extracted. The ranks of all the remaining nodes are decreased by one, yielding $k \le 3r+1$.

(5) Sort triplets: following the same argument as in (3), after the ordering $k \le 3r\text{-}1$ is valid again, completing the cycle.

After the first cycle, there is only one node in the queue and it lies in processor A_2, in agreement with $2 \le k \le 3r\text{-}1$. By induction, the relation therefore holds for every cycle. ■

2. 4 Multiple Input Systolic Priority Queue

In the section, we will describe the priority queue architecture for which the number of inputs is more than two. The new architecture can also deliver the node with best metric in constraint duration.

2.4.1 Multiple Input Systolic Priority Queue Algorithm (MISPQ)

The goal of MISPQ is to simultaneously insert N>2 new input metrics into the queue, and then deliver the best metric in the queue. The MISPQ is based on the operating principle of PESPQ, but it has some special circuit design to make the number of transmission gates fewer than that of PESPQ [Ref. 2]. Since it can support multiple input, we use it to implement the sequential decoding for 2/4 (3/6) convolutional codes.

The MISPQ architecture is shown in Fig. 1.2. In the figure, each column may be named as a slice. Processors in each slice belong to a group. The operations of systolic priority queue are usually considered slice by slice.

Definition 2.4.1:

Let $\boldsymbol{M_i}$ be the input metrics for $\boldsymbol{i}$=1, 2, …, N, where N is the number of inputs and the length of slices. Let $\boldsymbol{P_{i,j}}$ be the processor in a slice for $\boldsymbol{i}$=1, 2, …, and $\boldsymbol{j}$=1, 2, …, N, where $\boldsymbol{i}$ represents the $\boldsymbol{i}$-th slice and $\boldsymbol{j}$ the position of the processor in a slice. $\boldsymbol{P_{i,0}}$ stores the node with best metric in the $\boldsymbol{i}$-th slice.

<MISPQ algorithm>

Step 1. Insert N metrics $\boldsymbol{M_1}$, $\boldsymbol{M_2}$, …, $\boldsymbol{M_N}$ simultaneously into the queue and shift each metric at position $\boldsymbol{P_{i,j}}$ to its right.

Step 2. Rearrange metrics in each slice. Move the best metric in each slice to its local top position, and the position of other metrics is trivial.

Step 3. Extract the $\boldsymbol{P_{1,0}}$ from the queue, which is always the best metric among all. At the same time shift the metric on the top of each slice to its left.

Step 4. Rearrange metric in each slice of processors. Move the best metric in each slice to its local top position, and the position of other metrics is trivial.

A similar theorem as the one for PESPQ is derived to confirm that the systolic priority queue described can perform as desired.

Theorem 2.4.1:

For MISPQ, after any number of insertion or extraction operations, the k-th good metric is stored in some processor $P_{i,j}$, where $P_{1,0} \leqq P_{i,j} \leqq P_{k,0}$. For example, the best metric (k=1) will always reside at $P_{1,0}$ [Ref. 2].

Proof: By induction as follows. The first several steps can be easily checked by observations, so that one may assume that after m steps of operation the kth good metric resides in processor $P_{i,j}$ which satisfies $P_{1,0} \leq P_{i,j} \leq P_{k,0}$. Then one is to prove that it is still true at the (m+1)th step.

(a) If the (m+1)th step operation is an insertion, the best metric in the queue will still appear at $P_{1,0}$ for the kth good metric, where $k \geq 2$. First, if it resides in $P_{i,j}$ where $P_{1,0} \leq P_{i,j} \leq P_{k-1,0}$, then after the new N metrics being inserted this metric will still reside somewhere before $P_{k,0}$, and thus is a legal position (when k = 2 the case $P_{1,0} \leq P_{i,j} \leq P_{1,0}$ will not occur). Secondly, if it resides at $P_{i,j}$ with $P_{k-1,0} \leq P_{i,j} \leq P_{k,0}$, then after operation, this metric will reside at $P_{k,0}$. This is because the other metrics to be compared with this previous kth good metric all have ranks not lower than k, otherwise **Theorem 3** is violated at the mth step. Since the new rank of this metric will not be lower than k, this is a legal position again.

(b) If the (m+1)th operation is a deletion (extraction). The new best metric will appear at $P_{1,0}$. This is because the previous second good metric is originally at $P_{i,j}$ with $P_{1,0} \leq P_{i,j} \leq P_{2,0}$, after shifting all top metrics to the left the new best metric will appear somewhere among the first (N+1) positions. Now consider the position for the kth good metric. First, if the previous kth ($k \geq 3$) good metric resides at

some $P_{i,j}$ with $P_{1,0} < P_{i,j} \leq P_{k-1,0}$, then after operations it can still reside at a legal position. This is because $P_{1,0} < P_{i,j} < P_{k-1,0}$, thus is at a legal position for the (k-1)th good metric. Secondly, if the previous kth ($k \geq 3$) good metric resides at some position $P_{i,j}$ with $P_{k-1,0} < P_{i,j} \leq P_{k,0}$, then after operations that metric will appear at $P_{k-1,0}$. Because the metric to be compared with the previous kth good metric are located originally at some $P_{i,j}$ with $P_{k-1,0} < P_{i,j} \leq P_{k,0}$. Their ranks are all no less than k.

From (a) and (b), it can be seen that **Theorem 2.4.1** still holds at the (m+1)th step. By induction, **Theorem 2.4.1** is always true after any number of steps of the algorithm. ■

2.4.2 Type II Multiple Input Systolic Priority Queue

The MISPQ given in previous subsection is suitable for the queue with two inputs. However, if the number of inputs for queue is larger than 8, there is another structure [Ref. 2] which have smaller number of gates than cascading more than one MISPQ. This structure is named the type II MISPQ. The algorithm of type II is almost the same as MISPQ but it divided the slice in MISPQ into two parts to increase the sorting speed. Besides, it can also reduce the transmission gates [Ref 2].

Definition 2.4.3:

Divide the input M_1, M_2,…, M_N into M_1, M_2, …, $M_{N/2}$ and $M_{N/2+1}$, $M_{N/2+2}$, …, M_N. The slice $P_{i,j}$ is also divided into $T_{i,j}$ and $B_{i,j}$ and called subslice.

<Type II MISPQ algorithm>

Step 1. Insert N metrics M_1, M_2, …, M_N simultaneously into the queue and shift each

metric at position $T_{i,j}$ and $B_{i,j}$ to its right.

Step 2. Rearrange metric in each subslice of processors. Move the best metric in each subslice to its local top position $T_{i,0}$ and $B_{i,0}$, and the position of other metrics are trivial.

Step 3. Extract the $P_{1,0}$ from the queue, which is always the best metric among all. At the same time shift the metric on the top of each slice to its left according to the rule:

If (metric in $T_{i,0}$) ≥ (metric in $B_{i,0}$), and (metric in $T_{i-1,0}$) ≥ (metric in $B_{i-1,0}$) shift metric in $T_{i,0}$ to $T_{i-1,0}$

If (metric in $T_{i,0}$) ≥ (metric in $B_{i,0}$), and (metric in $T_{i-1,0}$) < (metric in $B_{i-1,0}$) shift metric in $T_{i,0}$ to $B_{i-1,0}$

If (metric in $T_{i,0}$) < (metric in $B_{i,0}$), and (metric in $T_{i-1,0}$) ≥ (metric in $B_{i-1,0}$) shift metric in $B_{i,0}$ to $T_{i-1,0}$

If (metric in $T_{i,0}$) < (metric in $B_{i,0}$), and (metric in $T_{i-1,0}$) < (metric in $B_{i-1,0}$) shift metric in $B_{i,0}$ to $B_{i-1,0}$

Step 4. Rearrange metric in each subslice of processors. Move the best metric in each slice to its local top position $T_{i,0}$ and $B_{i,0}$, and the position of other metrics are trivial.

It is found that the mechanism of type II is exactly the same as MISPQ. We introduce a theorem similar to theorem 2.4.1. According to the theorem, the best metric in type II is extracted for every cycle.

Theorem 2.4.2:

In every *i*th slice where I = 1, 2, …, if the processors Ti,0 and Bi,0 are viewed as a single processor $P_{i,0}$, and processors ($T_{i,1}$, $T_{i,2}$, …, $T_{i,N/2}$) and ($B_{i,1}$, $B_{i,2}$, …, $B_{i,N/2}$) are recombined into ($P_{i,1}$, $P_{i,2}$, …, $P_{i,N}$), then for a type II MISPQ, after any number of

insertion or deletion (extraction) operations, the *k*th good metric is stored in some processor $P_{i,j}$, where $P_{1,0} \leqq P_{i,j} \leqq P_{k,0}$.

Proof: As for theorem 2.4.1.[Ref. 2] ■

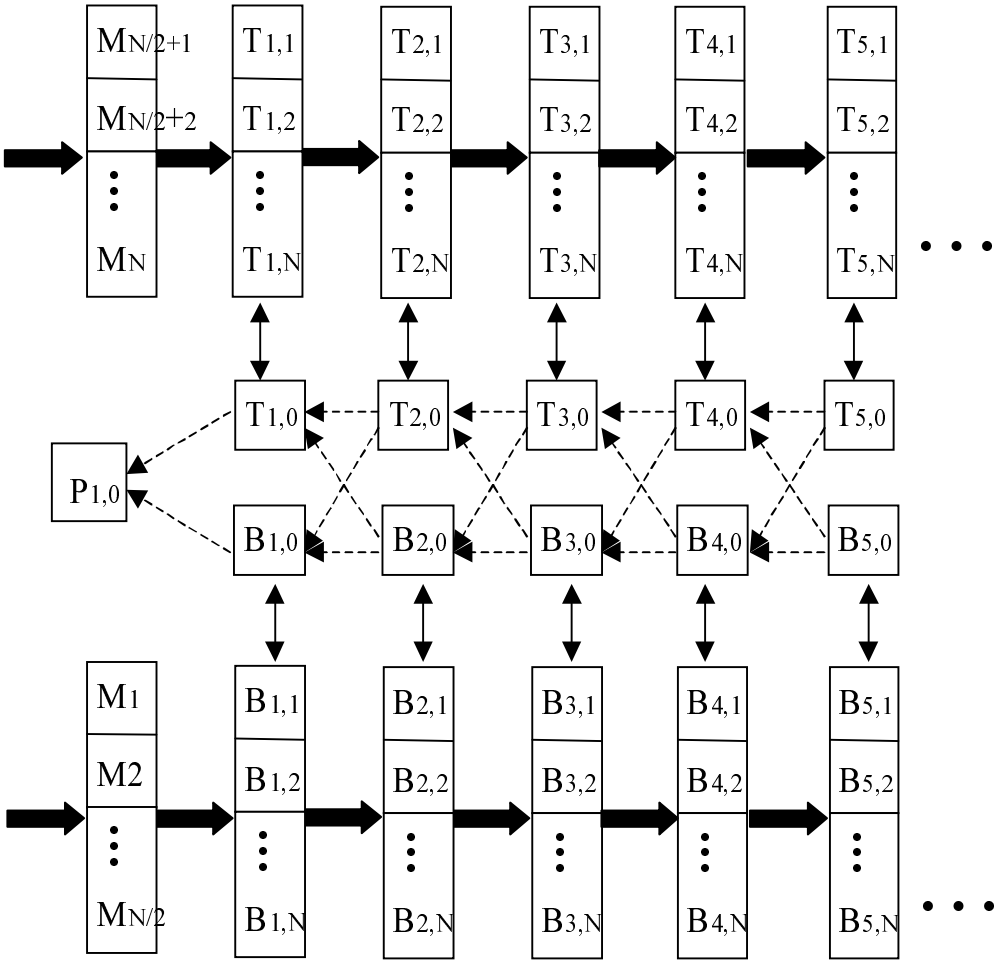

Fig. 2.2 type II MISPQ structure