# Matrix-based Nonblocking Routing Algorithm for Beneš Networks

Amitabha Chakrabarty, Martin Collier, Sourav Mukhopadhyay
School of Electronic Engineering
Dublin City University
Dublin 9, Ireland
Email: {amitabha, collierm, msourav}@eeng.dcu.ie

***Abstract*—This paper presents a nonblocking routing algorithm for Beneš networks. Such networks are of potential interest in implementing large scale optical cross-connects, but their complex routing algorithm limits their applications. We use a simple approach to determine the routing tags for a conflict free routing. Available methods proposed in the literature use computationally complex solutions to determine the routing tags. We propose a new approach for determining the paths through the network for all the requests coming to the inputs of the switch for unicast routing. Each switching stage has been represented by a set of sub-matrices and hence a conflict free routing is found. The correctness of the generated routing tags have been validated with mathematical method as well as with a simulator that can validate the correctness of the routing tags. This algorithm requires less complex practical implementation than the looping algorithm, making it viable for use in cross connect systems.**

**Keywords: Rearrangeable Network, Permutation, Interconnection Networks, Blocking, Routing Tags, Parallel Processing, Distributed Processing.**

## I. Introduction

Large scale computing strategy is the need for todays communication networks. Large scale VLSI (Very Large Scale Integration) implementation made parallel/distributed processing possible. For large systems, building the interconnection networks for processor-processor or processor-memory communication is of main concern. Flexibility and optimum cost along with faster communication is the greatest challenge in choosing an interconnection network. One network that emerged from Clos family [7] of networks with $2 \times 2$ switching elements and $N/2$ switching elements at each stage is the Beneš network [6], which can provide all the required characteristic needed for a parallel/distributed systems. The Beneš networks are a very well know nonblocking multistage interconnection networks [1], [3]. The Beneš networks are rearrangeably nonblocking networks [5], which means, paths of blocked calls are established by rerouting established calls through different paths. The Beneš network has $N = 2^n$ inputs and outputs and comprises $(2n - 1)$ stages of $2 \times 2$ switch elements. This network is a permutation network [6] because it can realize all *N!* possible patterns of input-output requests. Fig 1 shows a $16 \times 16$ Beneš Network. The Major focus of recent communication researches in electrical or optical domain is to increase the throughput of each switching ports rather than increasing number of ports. Increasing the port capacity after a certain boundary will introduce cross talk to the system [4]. In this paper we focus on Beneš network where to increase the capacity of the switch means increasing number of ports rather than port capacity.

### A. Related Works

The Beneš network is a long established method for establishing connections in connecting networks. Beneš networks found their use in shared memory multiprocessor systems [15], [1], telecommunication networks, TDMA systems [14]. It has been used as a permutation network in the middle of the switch fabric for routing packets from input queues to output. Researchers have given various routing algorithms for Beneš networks. In this section we will provide a general overview of the Beneš network routing algorithms from the literature. Along with others, Waksman [12] proposed a recursive algorithm for setting the switching element state in the Beneš network for uni-processor system. He showed that Beneš network is the shortest depth $2 \times 2$ rearrangeable network. The algorithm proposed by Opferman and Tsao-Wu [22], called the *looping algorithm*, works from the outer stage towards the center stage. It works by dividing the entire network into smaller networks and recursively setting paths in the smaller networks, there by setting the complete path. Later Andreson [23] provided an extended version of the looping algorithm for base $2^t$ networks. Nassimi and Sahni [16], [20] proposed a parallel self-routing method for a particular class of permutations. Nassimi and Sahni [21] proposed there way to implement Waksman's [12] approach in a parallel processing mode. To reduce the switch setting time in Beneš networks, Feng and Seo [9], [10] proposed *Inside-out routing* method. In this method they developed a new way to set up connecting paths for input/output request starting from the middle stage to the outward direction. Kim [13] showed that *Inside-out routing* method needs backtracking and even after back tracking its not fully blocking free. Lee [8], [11], proposed a non-recursive algorithm, where she divided the network in two parts: $NS1$ and $NS2$. Her algorithm works on a single stage of the network from left to right. Method proposed by Çam [17], uses the concept of *Balanced matrix* [18] and *2-colouring* to generate the routing tags for the input/output requests.

978-0-7695-3862-4/09 $26.00 © 2009 IEEE
DOI 10.1109/ComputationWorld.2009.72

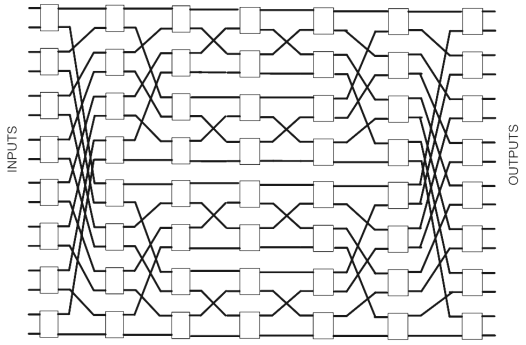


Fig. 1. $16 \times 16$ Beneš Network.

His proposed method is efficient provide that the PRAM (Parallel Random Access Machine) architecture [19] is used to implement the method. But there are still concerns surrounding the implementation of PRAM architecture for any practical system.

### *B.* Our Contribution

All the above methods cited in the related work section used methods that are computationally complex. In this paper we proposed a matrix based routing method to determine the routing tags. Rather than using collection of mathematical tools to generate the tags, we just start with assigning values in the target cells of a sub-matrix. The algorithm starts by setting a switching element in to straight through position at stage 0 for any one of its input for a particular output request. The algorithm continues this process for all the switching elements in that stage until we encounter any conflict of values in the cells. For any conflict, we change the switching elements setting by identifying cells sub-matrix. The algorithm then proceed to the next stage and continue setting up the switching element in that stage.

This paper is organized as follows, Section II gives the terminologies and notation used in this paper. Section III gives description of the proposed method followed by Section IV which explains the implementation part of the algorithm. Section V is the results accumulated from the implemented simulated followed by time analysis in Section VI. Future work is Section VII followed by conclusion in Section VIII.

## II. Preliminaries

This section provides notation used throughout the paper.

DEFINITION 1. (Input Permutation, $P$) The input permutation is a connection between input and output ports. In this paper, we will consider the mapping of an input to an output port as an element of a permutation. Let us assume that $P_{0:(N-1)} = x_i \in \{0 \dots (N-1)\}\ and\ x_i \neq x_j\ where\ 0 \leq (i,j) \leq (N-1)$. The mapping $P_i \rightarrow x_i$ suggest that input $i$ is requesting output port $x_i$.

DEFINITION 2. (Switching Element, $SE$) Switching elements is the basic building block of a switching network. In a Beneš networks total number of switching elements are $(2logN-1)\times N/2$. With a matrix notation a switching element can be identified with its position in the switch with a notation $[i,j]$ , $0 \leq i \leq (N/2)-1$.

DEFINITION 3. (Sub-matrix) At each stage for the switching network number of sub-matrices is $2^i$, where $0 \leq i \leq (logN-2)$. We determine the switching element setting at each stage by putting values 0 or 1 in the cells of the sub-matrices by rearranging the blocked calls at each stage.

DEFINITION 4. (Blocking State) Each rows and columns in all the sub-matrices are allowed to have only one 0 and one 1. In case of a scenario where a sub-matrix end up having more than one $0's$ or $1's$ in the same row or column, in this paper we call this as a blocking state for the sub-matrix.

DEFINITION 5. (Balanced State) One common characteristic of Beneš networks is that the binary column vectors of input matrix at each stage of the network is balanced if the routing is blocking free, even though the sequence of elements can't have any order between them. Beneš network has two parts in the network, one follows distributed routing and the other is bit-controlled. One feature of this is that column matrices of each part create two different balanced matrices but concatenation of these two balanced matrices are not balanced. These two different balanced matrices creates a balanced state in the network, which is a condition of a conflict free routing. If $C_i$, where $0 \leq i \leq (logN-2)$, are column vectors for the distributed part of the Beneš network then for a conflict free routing each $C_i$ has to be balanced as well as the matrix created from the concatenation of $C_i's$.

For example, If $C_0$ and $C_1$ are two binary column vectors which are given by,

$$\mathbf{C_0} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \mathbf{C_1} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \tag{1}$$

If $C_0$ and $C_1$ are individually balanced then concatenation of $C_0$ and $C_1$ has to be balance for routing tags to be conflict

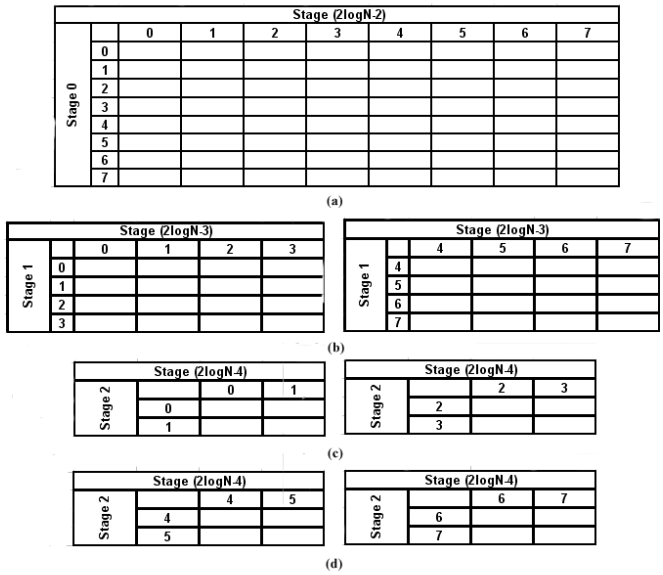


Fig. 2. (a)-(d) Sub-matrices format for a $16 \times 16$ Beneš Network.

free.

## III. Proposed Method for Routing in Beneš Networks

In this section, we describe our method in *two* phases.

### A. Phase 1(Setting the Switching Elements)

**Input:** Input Permutation $P_{0:(N-1)}$.

**Output:** Routing tags for stage $i$, $0 \leq i \leq (logN - 2)$ .

**Step 1:** Create $2^i$ sub-matrices where $0 \leq i \leq (logN - 2)$ and $i$ identifies the stages of the network.

**Step 2:** At stage $i$ where $0 \leq i \leq (logN - 2)$, index the rows of sub-matrices from 0 to $N/2$ sequentially. Create row sized columns for each sub-matrices. Each row takes values for two inputs for the same switching element (SE) at the input stage and each column identifies requested switching element port for specific inputs.

**Step 3:** At state 0, start with input 0, identify the requested output switching element. In the row of input port put a 0 at the column for requested switching element. For input 1, put a 1 at the same row but in the column that points to the requested output switching element.

**Step 4:** Continue $Step$ 3 for remaining input/output requests of the permutation. For each row and column only one 0 and 1 can be used. Fig 2 shows the sub-matrices format for a $16 \times 16$ Beneš network. In case of a situation with multiple $0's$ or $1's$ the sub-matrix enters in to a a blocking state. To break the blocking state goto $phase$ 2.

### B. Phase 2 (Rearrangements Inside Sub-matrices)

**Input:** Blocking State in a sub-matrix.

**Output:** Rearrangement of $0's$ and $1's$ to unblock a blocked request.

**Step 1:** In the blocked row, rearrange the values between cells in that row.

**Step 2:** In case of any new blocking introduced because of last rearrange, change the value of the column other than the one affected by the rearrange . If there is no blocking, then the request is unblocked, goto $Phase$ 1 to set switching elements for remaining requests else goto $step$ 1.

### C. Example

Let $N = 16$ and a random permutation $P_{(0:15)} =$ ( 5 14 11 12 9 2 8 1 13 6 0 10 3 4 15 7 ). Since the switch size is $16 \times 16$, number of sub-matrices for stage 0 is one with a size of $8 \times 8$. Similarly for stage 1 its two with size $4 \times 4$ and for stage 2 its four having size $2 \times 2$. According to the given permutation, input 0 corresponding to the input switching element 0 goes to output 5 which is in output stage switching element 2. Put a 0 at the $[0, 2]$ position on the sub-matrix for stage 0. Similarly for input 1 requested output is 14 which corresponds to output switching element 7. Since we can only use 0 or 1 once in a single row or column and we already used 0 at position $[0, 2]$, so put a 1 at position $[0, 7]$. Continue the above process for remaining request in permutation till we reach at row 5, which corresponds to input 10 and 11. Requests for these two inputs are 0 and 10 respectively. Since in row 3 we have a 0 at position $[3, 0]$, we put a 0 at position $[5, 0]$ and a 1 at position $[5, 5]$. As a result of using 1 at position $[5, 5]$, the sub-matrix enters into

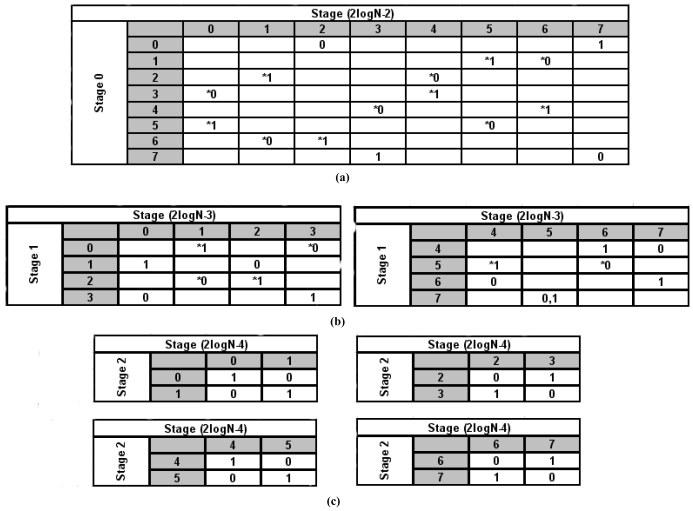


Fig. 3. (a)-(c) Sub-matrices status after execution of the algorithm for a $16 \times 16$ Beneš Network. The sign * indicates interchange.

a blocking state. To overcome this state change the value at at $[5, 5]$ from 1 to 0 and change the value at $[5, 0]$ from 0 to 1. Making these interchange keeps the sub-matrix still in to a blocking state because multiple 1 in column 0. Change the value of $[3, 0]$ to 0 and continue the hole process till the sub-matrix is out of blocking state. After finishing the first stage, the values corresponding to all the input is the values for setting the switching elements at stage 0.

For remaining of the distributed part continue the above process to generate the routing tags. Fig 3 shows the status of the sub-matrices after execution of phase 1 and 2 of the proposed algorithm.

**Algorithm 1** : Routing Algorithm For Beneš Networks.

INPUT: $P_{0:(N-1)}$
OUTPUT: Conflict Free Routing Tags

1: **for** $i \leftarrow 0$ **to** $(N/2) - 1$ **do**
2: **for** $j \leftarrow 0$ **to** 1 **do**
3: find Int $K \in \{0, 1\}$, $K \neq I_j, O_{i_j}$
4: **if** $K = NULL$ **then**
5: $K \leftarrow 0$
6: **end if**
7: $(I_j, O_{i_j}) \leftarrow K$
8: **end for**
9: **end for**

## IV. Procedures and Implementation

This section describes the pseudocodes used to implement the method proposed in section 3. Algorithm 1 describes the switching elements setting when there is no conflict inside any of the sub-matrices. In case of any conflict Algorithm 1 calls Algorithm 2.

In Algorithm 1, among the two for loops first one is for each input switching elements, and the second one is for assigning vales 0 or 1 in each row and column. Variable $K$ becomes $NULL$ when the target row and column has no cell which has a value of 0 or 1. Variables $I_i$ and $O_{i_j}$ indicates the target row and column respectively. Step 7 of the procedure assigns values in the target cell and in case of a blocking state in the sub-matrix, Algorithm1 calls Algorithm 2. In Procedure 2 we start with blocked row identified as $K$ and implement the interchange process to unblock a blocked call. Line 2 checks for multiple $0's$ and line 3 replaces one with a 1. Line 12 and 13 executes the same functionality for multiple 1. After replacing multiple values in a row, next step is to execute functions to replace multiple entries in a column. This process continues until all the blocking states have been removed. We implemented our simulator following the above procedures. The results presented in next section are based on the outcome of simulator for various size of networks with full load traffic.

## V. Simulation Results

This section will provide the results that have been extracted from the simulator. At first we will discuss about the generated routing tags from this method and also test its validity with mathematical prove as well as with simulator. The output of the second simulator will be tested with the original input permutation to test the validity of the routing tags. This will prove that our proposed method can determine conflict free paths for each and every input/output requests.

### A. Routing Tags and its Validity

For a $16 \times 16$ network we generated routing tags using our simulator for randomly generated permutations. Among them a permutation is

**Algorithm 2** Rearranging Inside Conflicting Sub-matrix

INPUT: Blocking State Inside a Sub-matrix
OUTPUT: Balanced State In The Sub-matrix
1: $K \leftarrow Blocked\ Row$
2: **if** for column $C$, $(K, C) = 0$ **then**
3: $(K, C) \leftarrow 1$
4: in column $C$, search $(K', C) = 1$
5: $(K', C) \leftarrow 0$
6: $K \leftarrow K'$
7: in row $K$, search $(K, C') = 0$
8: $(K, C') \leftarrow 1$
9: $C \leftarrow C'$
10: goto 4, else request is unblocked
11: **else** find column $C$, where $(K, C) = 1$
12: $(K, C) \leftarrow 0$
13: in $C$, search $(K', C) = 0$
14: $(K', C) = 1$
15: in $C'$, search $(K, C') = 1$
16: $(K, C') = 0$
17: $C \leftarrow C'$
18: goto 12, else request is unblocked
19: **end if**

$P_{(0:15)}$ = ( 1 9 2 5 15 10 7 8 13 0 6 3 11 12 4 14 ). Matrix $C_T$ shows generated routing tags for the given permutation, three column vectors $C_0$, $C_1$ and $C_2$ are taken from matrix $C_T$.As it can be seen that all the column vectors are balanced. The concatenated matrix $C$ is made of concatenating vectors $C_0$,$C_1$ and $C_2$. It is clear that matrix $C$ is a balanced matrix. It can be proved that the concatenated matrix $C$ with $(logN - 1)$ columns will always be balanced. Each columns of $C$ has the elements taken from all the sub-matrices created for the network. Since there can not be any reparations of $1's$ or $0's$ in same row or column so each column vector will be balanced.

As a result concatenation of $(logN - 1)$ balanced column vectors will give a $N \times (logN - 1)$ balanced matrix. The binary output matrix $C_O$ is itself a balance matrix since it a one-to-one request permutation. So combination of $C \bullet C_O = C_T$ will create a balanced state in the network which is the condition for a conflict free routing, even though the combined matrix is not balanced.The sign $(\bullet)$ indicates a concatenation.To further confirm our statement, the generated routing tags are tested in a simulator. This simulator takes the generated routing tags as inputs, generate the status matrix which is the status of the switching elements in an $N \times N$ switch. We then generate the input permutation from this status matrix. Fig 4 shows the generated routing tags, status matrix and input permutation for a $16 \times 16$ network. The generated input permutation matches with the reference permutation.

## VI. TIME ANALYSIS

Our proposed method has a setup time of $O(NlogN)$ for a uni-processor system, which is similar to the looping algorithm. One major consideration with the looping algorithm is the real system implementation. It is quite difficult to build a system that can keep track of internal permutation which is required at each stages and each subnetworks of the network. The circuit complexities may be too high for looping to be a practical choice. But with our proposed method less complex circuitry would be able to generate the routing tags for the network. So even though both the methods have equal setup time complexity, our method would be more useful for practical implementation.

## VII. FUTURE WORK

We believe that the proposed method opens few future opportunities to be explored. Among them the first would be to map this method for various available parallel machine architecture and compare the timing factors with existing methods. One other investigation could be to look at the prospect of rearranging chains without breaking the input/output communication paths. Establishing a new chain without breaking the connection is still a major concerns for rearrangeable networks. With necessary modification we might be able to determine needed mechanism for establishing a chain without breaking existing circuit.

$$\mathbf{C_T} = \begin{bmatrix} 0000001 \\ 1001001 \\ 0100010 \\ 1100101 \\ 0011111 \\ 1111010 \\ 1010111 \\ 0111000 \\ 0101101 \\ 1000000 \\ 0010110 \\ 1110011 \\ 0001011 \\ 1011100 \\ 0110100 \\ 1101110 \end{bmatrix}$$

$$\mathbf{C_0} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \quad \mathbf{C_1} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{C_2} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

$$\mathbf{C} = \mathbf{C_0} \bullet \mathbf{C_1} \bullet \mathbf{C_2} = \begin{bmatrix} 000 \\ 100 \\ 010 \\ 110 \\ 001 \\ 111 \\ 101 \\ 011 \\ 010 \\ 100 \\ 001 \\ 111 \\ 000 \\ 101 \\ 011 \\ 110 \end{bmatrix}$$

| | | |
|---|---|---|
| **0000001** | | **0 :-> 1** |
| **1001001** | | **1 :-> 9** |
| **0100010** | | **2 :-> 2** |
| **1100101** | | **3 :-> 5** |
| **0011111** | | **4 :-> 15** |
| **1111010** | | **5 :-> 10** |
| **1010111** | | **6 :-> 7** |
| **0111000** | | **7 :-> 8** |
| **0101101** | **0000001** | **8 :-> 13** |
| **1000000** | **0011010** | **9 :-> 0** |
| **0010110** | **0100010** | **10 :-> 6** |
| **1110011** | **1001110** | **11 :-> 3** |
| **0001011** | **0001000** | **12 :-> 11** |
| **1011100** | **0100011** | **13 :-> 12** |
| **0110100** | **0000101** | **14 :-> 4** |
| **1101110** | **0011101** | **15 :-> 14** |
| **(a)** | **(b)** | **(c)** |

Fig. 4. (a) Generated Routing Tags. (b) Status matrix. (c) Input Permutation from Status Matrix.

## VIII. Conclusion

In this paper, we presented a routing algorithm that uses a simple method to determine conflict free paths for input/output requests in Beneš Networks. We proved the validity of the generated routing tags both mathematically as well as with simulator results. This method has the same time complexity as the looping algorithm but our method is less complex to implement. It sets switching elements stage by stage which means there is no need to broadcast switching elements setting to every stage. It thus constitutes a viable method to implement large scale optical cross connect as it will be needed in tomorrows Internet.

## References


[1] Sandeep Sharma, P.K.Bansal, and Karanjeet Singh Kahlon. *ON A Class Of Multistage Interconnection Network In Parallel Pprocessing*. International Journal of Computer Science and Network Security, Vol.8, No.5, May 2008.

[2] Frank K. Hwang, and Wen-Dar Lin, Vadim Lioubimov. *On Noninterruptive Rearrangeable Networks*. IEEE/ACM Trans. Networking, Vol. 14, No.5, October 2006.

[3] Fong-Chih Shao, and A. Yavuz Oru. *Efficient Nonblocking Switching Networks for Interprocessor Communications in Multiprocessor Systems*.IEEE Trans. on Parallel and Distributed Systems, Vol. 6, No.2, February 1995.

[4] C.H. von Helmolt *Electrical and acoustical crosstalk in integrated optical strip waveguide devices*. IEEE Electronics Letters. Volume 18, No. 22 , 1982.

[5] Andrzej Jajszczyk. *Nonblocking, Repackable, and Rearrangeable Clos Networks: Fifty Years of the Theory Evolution*. IEEE Communications Magazine, October 2003.

[6] E. Benes. *Mathematical Theory of Connecting Networks and Telephone Traffic*. New York: Academic Press, 1965.

[7] C. Clos. *A Study of Non-Blocking Switching Networks*. Bell System Technical J., vol. 32, pp. 406-424, 1953.

[8] K. Y. Lee. *On the rearrangeabllity of* $2(log_2 N-1)$ *stage permutation networks*. IEEE Trans. Comput, vol. C-34, no. 5, pp 412-425, May 1985.

[9] S.-W. Seo, T.-Y. Feng, and H.-I. Lee. *Permutation Realizability and Fault Tolerance Property of the Inside-Out Routing Algorithm*.IEEE Trans. Parallel and Distributed Systems, vol. 10 ,no. 9 , pp. 946-957, 1999.

[10] T.-Y. Feng and S.-W. Seo. *A New Routing Algorithm for a Class of Rearrangeable Networks*. IEEE Trans. Computers, vol. 43, no. 11, pp. 1270-1280, 1994.

[11] K. Y. Lee. *A new Benes network controlalgorithm and Parallel Permutation Algorithm*. IEEE Trans. Comput. ,vol. C-30 , no. 5, pp. 157-161, May 1981.

[12] A. Waksman. *A Permutation Network*. J. ACM, vol. 15, no. 1, pp. 159-163, Jan. 1968.

[13] M.K. Kim, H. Yoon, and S.R. Maeng.*On the Correctness of Inside-Out Routing Algorithm*. IEEE Trans. Computers, vol. 46, no. 7, pp. 820-823, July 1997.

[14] S. Keshav.*An Engineering Approach to Compter Networking*. Addison-Wesley, 1997.

[15] F.T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. San Mateo, Calif.Morgan Kaufmann, 1992.

[16] D. Nassimi and S. Sahni. *A self-routing Benes network and Parallel Permutation Algorithm*. IEEE Trans. Comput., vol. C-30, no.5, pp.332-340, May 1981.

[17] Hasan Çam and Jose A.B. Fortes.*Work-Efficient Routing Algorithms for Rearrangeable Symmetrical Networks*. IEEE Trans. on Paraller and Distributed Systems, Vol.10, No.7, July 1999.

[18] N. Linial and M. Tarsi, *Interpolation between Bases and the Shuffle-Exchange Network*. Europian J. Combinatorics, vol. 10, pp. 29-39, 1989.

[19] R. Cypher, J. L. C. Sanz, L. Snyder. *An EREW PRAM Algorithm for Image Component Labeling*. IEEE Trans On Pattern Analysis Aand Machine Intelligence.Vol .11, No. 3. March 1989.

[20] D. Nassimi and S. Sahni. *A self-routing Benes network*. Proceedings of the 7th annual symposium on Computer Architecture. La Baule, United States Rep. pp: 190 - 195, May 1980.

[21] D. Nassimi and S. Sahni.*Parallel Algorithms to Set Up the Benes Permutation Network*.IEEE Trans. Comput., Vol. c-3 1, No. 2, February 1982

[22] D.C. Opferman and N.T. Tsao-Wu. *On a Class of Rearrangeable Switching Networks, Part I: ControlAlgorithm* Bell System Technical J., vol. 50, pp. 1,579-1,600, 1971.

[23] S. Andresen. *The looping algorithm extended to base* $2^t$ *rearrangeable switching networks*. IEEE Trans.Commun., vol. COM-25, no. 10, pp.1057-1063, Oct. 1977.