On Self Routing in Beneš and Shuffle Exchange Networks*

Rajendra Boppana C. S. Raghavendra
Dept. of Electrical Engineering-Systems
University of Southern California
Los Angeles, CA 90089

Abstract

A self routing algorithm for passing Linear class of permutations in Beneš, π and $(2n-1)$-stage shuffle exchange networks of $N = 2^n$ inputs/outputs is presented. In these networks, switches in the first $(n-1)$ stages are set by comparing the destination tags of the inputs to the switch; switches in the remaining stages are set by the self routing Ω algorithm. Thus, the total time required for routing any Linear permutation is $O(n)$, same as the network delay time. The algorithm also routes Ω^{-1} permutations in Beneš and Ω permutations in π network trivially. The class of permutations that are routable by the algorithm is much richer than the class of Linear permutations. This algorithm routes all possible permutations for 4 input/output Beneš network $B(2)$ (same as 3-stage shuffle exchange network) and π-network, since all the permutations are in the Linear Class.

1 Introduction

Typically, a parallel computer consists of a number of processors and an interconnection network for exchange of information between them as well as with memory modules. Considering a processor/memory network model, any processor should be able to communicate with any memory module which is called full access. To support SIMD type computations, ideally we would like the network to be able to perform all the permutations that allow simultaneous use of the memory modules. Such capabilities exist in crossbar networks and networks that are rearrangeable, for example the Beneš network.

We view parallel computing as computation steps—during which time some or all of the processors are busy computing, and communication steps—at which some permutation function is set up by the network to allow data exchanges. If the underlying network can not support a required permutation function then it has to be realized in multiple steps. The advantage with a rearrangeable network is that any permutation can be realized in one communication step. Further, if they are built using smaller switches such as 2×2, then they are relatively cheaper than crossbar networks. Therefore rearrangeable networks are used in some parallel computer implementations (e.g. GF-11 [1]).

A well known rearrangeable network is the Beneš network [2] which is built in a recursive manner using 2×2 switches, and is shown in figure 1. In such networks, it takes some time to set up the switches to realize a given arbitrary permutation. For an $N = 2^n$ inputs and outputs Beneš network, determining the switch settings to realize an arbitrary permutation takes $O(N \log N)$ time on a uniprocessor computer[7]. If the required permutations change frequently while computing a problem, the communication time may become a bottleneck. An approach to solve this problem is to compute the switch settings for a given permutation using a parallel computer with N PE's. A separate network with static links between the PE's in the parallel computer under consideration could be used for this computation as suggested by Nassimi and Sahni[6]. Alternatively, the Beneš network itself can be set to realize perfect shuffle permutation easily, to convert the parallel computer under consideration to a perfect shuffle computer and determine the switch settings in $O(n^5)$ time using the algorithm proposed by Nassimi and Sahni[5]. However, it still takes considerable amount of time to realize a permutaion compared to the propogation delay $O(n)$.

We are interested in developing fast self-routing algorithms for many useful permutations required in parallel processing, if not for all the $N!$ permutations. Due to the nature of techniques used in developing parallel algorithms, the permutaitons required are generally nice and regular and can be expressed as algebraic functions. Some work was done on developing self-routing algorithms for classes of permutations, in particular Bit-Permute-complement (BPC) by Nassimi and Sahni[6]. They also prove that their algorithm routes the Lenfant's FUB families[3].

In this paper we develop self-routing algorithms for the Linear Class ($\mathcal{L}$) of permutations. The algorithm is very simple and routes many other classes of permutations as well. We consider Beneš network as well as the π network of Yew and Lawrie[8] and $(2n-1)$-stage shuffle exchange network. The results include simple routing algorithms for the classes $\mathcal{L}$ (we extend this class with complements of bits), Ω, and Ω^{-1} on all these networks. For other permutations one can use a general looping type algorithm or break it into multiple simpler permutations.

2 Routing in Beneš Network

We will use I to represent any source and O to represent its destination tag. All binary additions in this paper are modulo 2.

*This research is supported by the NSF Presidential Young Investigator Award No. MIP 8452003, DARPA/ARO Contract No. DAAG 29-84-k-0066, ONR Contract No. N00014-86-k-0602.

International Conference on Parallel Processing, pp. 196-200, 1988

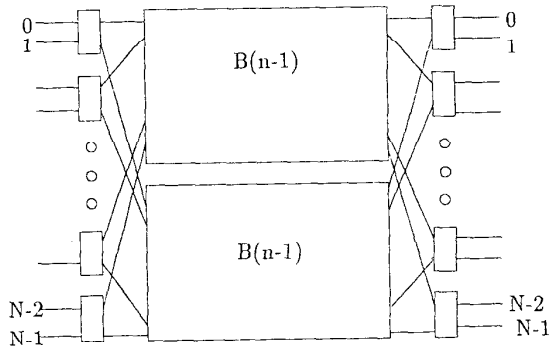


Figure 1: 2^n input/output Beneš network $B(n)$.

Definition 1 *A permutation is said to be a linear permutation*[4] *if for all input I (whose binary representation is* $(I_n\ I_{n-1}\ \ldots\ I_1)$*) and output O (whose binary representation is* $(O_n\ O_{n-1}\ \ldots\ O_1)$*) pair there exists a non singular binary matrix* $Q_{n\times n}$ *that satisfies 1.*

$$O^T = Q \times I^T \tag{1}$$

Definition 2 *Let* $I' = (I_n\ I_{n-1}\ \ldots\ I_1, 1)$*. A permutation is a Linear-Complement (*$\mathcal{LC}$*) permutation if there exists a binary matrix* $P_{n\times n+1}$ *where the submatrix of P formed by taking first n columns is non singular, such that every (I,O) pair satisfies the equation 2.*

$$O^T = P \times I'^T \tag{2}$$

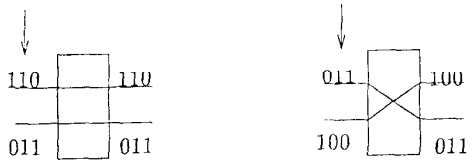
With the definition given above, $\mathcal{LC}$ contains $\mathcal{BPC}$. Throughout this paper we will assume that the number of inputs/outputs to the interconnection network is $N = 2^n$. We will denote linear-complement, omega and inverse omega permutations on N inputs in compact form as $\mathcal{LC}(n)$, $\Omega(n)$ and $\Omega^{-1}(n)$ respectively. And $\mathcal{B}(n)$ denotes Beneš network with N inputs/outputs.

2.1 Routing Algorithm

Let the output lines of a switch be numbered as '0' and '1' for upper and lower outputs respectively. Each input line to a switch will have a routing bit. An input line to a switch is connected to the output line of the switch indicated by its routing bit. If the bit is '1' then that input is connected to the lower output of the switch otherwise, it is connected to the upper output of the switch. Routing of $\mathcal{LC}$ permutations in Beneš network is given by the following algorithm.

Algorithm 1 For the first $(n-1)$ stages, an input line to a switch in stage $i, 1 \le i \le (n-1)$ will have i-th bit of its destination tag as its routing bit. For the next n stages, an input line to a switch in stage $j, n \le j \le (2n-1)$ will have $(2n-j)$-th bit of its destination tag as its routing bit. For the first $(n-1)$ stages, switches are set up such that input line with smaller destination tag value is routed according to its routing bit. For the next n stages switches, are set up such that both the inputs are routed according to their routing bits. ∎

In first $(n-1)$ stages, conflicts are resolved by giving priority to one of the input lines. This algorithm is different from that of Nassimi and Sahni's[6] since in case of conflict in setting up a switch, their algorithm gives priority to the top input line, whereas our algorithm gives priority to the input line with smaller destination tag value. Consider figure 2(a) with destination tags for its inputs as shown. Let the bit indicated by the arrow be the routing bit. In this case, routing bit for both the inputs is '1' so there is a conflict. This is resolved by comparing the destination tags and giving priority for the input with smaller destination tag value, which in this case is the lower input. The other input line is automatically routed to the remaining output line. In figure 2(b) routing bits for both inputs are different so they get what they want and the switch is set as shown.



(a) (b)

Figure 2: An example showing switch settings done by the algorithm.

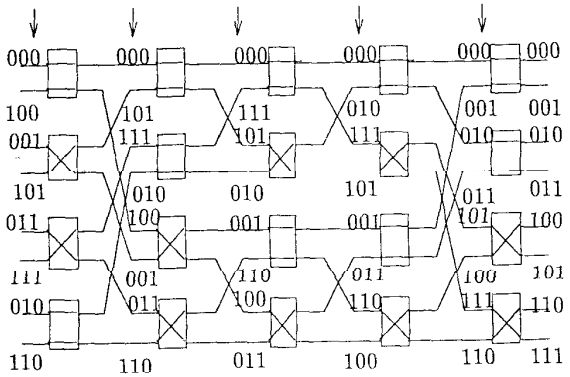


Figure 3: Routing a $\mathcal{LC}$ permutation in Beneš using the algorithm proposed

A complete example of this routing scheme is given in figure 3. Destination tags for each input line to a switch are given in the binary form. Routing bit for each stage is indicated by an arrow. This permutation is not routable by Nassimi and Sahni's (see figure 4) algorithm. The $\mathcal{LC}$ permutation given in the figures 3 and 4 has the functional form given below.

$$O_3 = I_1;\ O_2 = I_3;\ O_1 = I_3 + I_2$$

In the first stage (figure 3), routing bit is same for both the inputs to a switch. Hence switches in the first stage are set up such that input with smaller destination tag is routed correctly, which in this case are top input lines. After the

first stage of routing, there exists $\mathcal{LC}$ permutation between O_3, O_2 of destination tag and I_2, I_1 of input line, for both

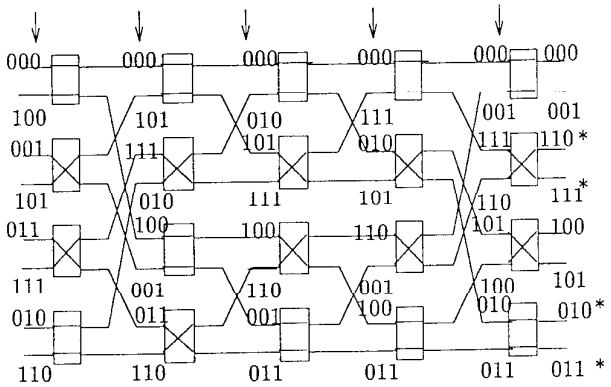


Figure 4: Routing an $\mathcal{LC}$ in Beneš using Nassimi and Sahni's algorithm fails. Incorrectly routed inputs are indicated by an asterisk.

top 4×4 Beneš network $\mathcal{B}(2)$ given as,

$$O_3 = I_2 + I_1; \quad O_2 = I_2$$

and bottom $\mathcal{B}(2)$ given as,

$$O_3 = 1 + I_2 + I_1; \quad O_2 = I_2$$

There exists conflict in setting up switches in the second stage of the network as well. For top most and bottom most switches in the second stage top input line has a smaller destination tag value, so these switches are set to route top input line correctly. For the other two switches bottom input lines have smaller destination tag value, hence, those switches are set to route bottom input lines correctly. Conflict exists only in the first two stages of the network. Last 3 stages are routed without any conflicts as given by the algorithm.

2.2 Proof of Correctness

Theorem 1 *Any $\mathcal{LC}(n)$ permutation is routable by the routing algorithm 1, in $\mathcal{B}(n)$.*

Proof: We will use the fact that stages $2, \ldots, 2n-2$ of $\mathcal{B}(n)$ are just two $\mathcal{B}(n-1)$ networks, to prove the theorem by induction. To do this we need to show that after first stage of routing, the resulting permutation between most significant $(n-1)$ bits of the destination tag to an input of $\mathcal{B}(n-1)$ is still an $\mathcal{LC}(n-1)$ permutation.

More formally, this is true for $n = 1$. Let it be true for all $m < n$. Now consider the following lemma.

Lemma 1 *After one stage of routing of an $\mathcal{LC}(n)$ permutation using the algorithm 1, for any input-output pair I and O, the permutation between $(O_n, \ldots, O_2)$ and $(I_{n-1}, \ldots, I_1)$ for the top and bottom Beneš networks for 2^{n-1} inputs/outputs belong to $\mathcal{LC}$(n-1).*

Proof for the lemma: Inputs to a switch differ only in bit I_1. So depending on whether the equation for routing bit O_1 contains I_1 or not, the routing tags of the inputs to a switch are different or are same. Consider the first case; the equation for O_1 will be of the form $O_1 = I_1 + LF_1$, where LF_1 is independent of I_1. Since each input is routed according to its routing bit because there are no conflicts, the equation for O_1 after exchange is given as $O_1 = I_1$. So the effect of exchange is like substituting $I_1 + LF_1$ in all occurrences of I_1 in the equations for $O_n, \ldots, O_1$. Since an inverse shuffle is performed after exchange, all the top outputs of the switches go to the top Beneš network $\mathcal{B}(n-1)$ and all bottom outputs of the switches go to bottom Beneš network $\mathcal{B}(n-1)$. So substituting $I_1 = 0(1)$ in the equations for bits $O_n, \ldots, O_2$ of the routing tags of the inputs routed to top(bottom) $\mathcal{B}(n-1)$ we get $\mathcal{LC}(n-1)$ permutation as desired. In the second case, the equation for O_1 will be of the form, $O_1 = LF_1$, where LF_1 is independent of I_1. Let k be the most significant bit in which two destinations differ. Then the equation for O_k contains I_1 and is given as $O_k = I_1 + LF_k$, LF_k is independent of I_1. The algorithm routes inputs such that input with $O_1 = O_k$ is routed to top ouput line of the switch and the other input to the bottm ouput line of the switch. So after the exchange operation $O_1 = I_1 + O_k$. So the net effect is equivalent to substituting $I_1 + LF_1 + LF_k$ in all the occurrences of I_1 in the equations for $O_n, \ldots, O_1$. Since an inverse shuffle is performed after exchange, as in the previous case we get $\mathcal{LC}(n-1)$ permutation between $O_n, \ldots, O_2$ bits of the routing tags and inputs $I_{n-1}, \ldots, I_1$ of the top and bottom $\mathcal{B}(n-1)$ networks. ∎

From the above lemma $\mathcal{LC}$ (n) is routed in the first stage of $\mathcal{B}(n)$ such that there exists $\mathcal{LC}(n-1)$ permutation between $O_n, \cdots, O_2$ and the inputs of $\mathcal{B}(n-1)$. Since this is correctly routed by induction hypothesis, after $(2n-2)$ stages all the outputs are in the correct place as far as first $n-1$ bits are concerned. This means two destinations which differ only in the last bit of their destination do not exist in the same $\mathcal{B}(n-1)$. A shuffle and exchange will route these inputs to the correct places. ∎

3 Routing in Shuffle Exchange Networks

We will modify the routing algorithm to route $\mathcal{LC}$ permutations in π–network. A π–network is a cascade of two Ω networks [8].

3.1 Routing Algorithm

Algorithm 2 For the first n stages of the *pi*–network, an input to a switch in stage $i, 1 \leq i \leq n$, will have $(n-i+1)$–th bit of its destination tag as the routing bit. Routing is done as follows. First the destination tags are bit reversed and then compared. The smaller one will be routed according to its routing bit as before. For the next n stages of the network we use the standard Ω self–routing algorithm. ∎

A complete example is given in figure 5. Routing bit in each stage is indicated by an arrow. This permutation is not routable by the self routing algorithm given in [8].

Consider second switch from top in stage 1 of figure 5. Both inputs have the same routing bit '1'. But upper input has destination tag with smaller value when compared to that of lower one after bit reversal of the destination tags. Hence upper input is routed to the lower output of the switch. But in the case of bottom most switch in the first stage lower input has smaller destination tag value after bit reversal. So, that switch is set such that lower input is routed according to its routing bit which is lower output of the switch.

3.2 Proof of Correctness

We need the following lemmas, to prove that the algorithm works correctly.

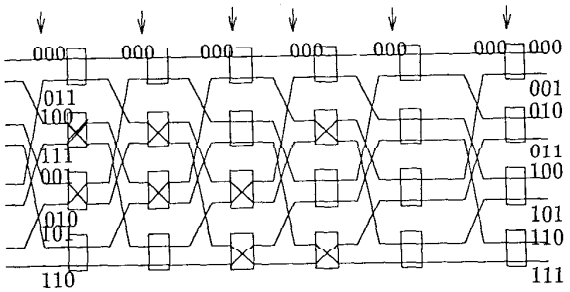


Figure 5: Routing an $\mathcal{LC}$ permutation in π-network using the algorithm proposed

Lemma 2 *If a permutation is $\mathcal{LC}$ permutation then after a shuffle on the input bits, the resulting permutation is still $\mathcal{LC}$.*

Proof is obvious, hence omitted. ∎

Lemma 3 *If a permutation is $\mathcal{LC}$ permutation then after performing an exchange operation on the inputs using the algorithm 2 the resulting permutation is still $\mathcal{LC}$.*

Proof for this lemma follows very closely that of lemma 1. Crucial part of the proof is showing that for the first n stages the algorithm performs exchange operation such that routing bit O_i is set to I_{n-i+1} if the equation for O_i contains I_{n-i+1}, otherwise to $I_{n-i+1} + O_j$ for some $j < i$ as specified by the algorithm. ∎

In the case of Beneš network we noted that two input lines to a switch in stage 1 differ only in I_1. However, in the case of the π network we shall take into account the fact that a shuffle was performed before the exchange operation hence I_n becomes I_1. So the two input lines to a switch in the first stage of a π network differ only in I_n. In the first stage, algorithm 2 will route using O_n as the routing bit. So there will not be any conflicts if O_n contains I_n. Proceeding in this manner it is easy to see that for the first n stages there will not be any conflicts if the routing bit O_{n-i+1} contains I_{n-i+1}.

Lemma 4 *Routing an $\mathcal{LC}$ permutation using algorithm 2 will always assure that at any stage i, $1 \leq i \leq n$, the destination tags for the inputs of a switch will differ atleast in one of the bits O_{n-i+1} through O_1.*

Proof: This is true for stage 1 since $\mathcal{LC}$ is a bijection. After $(m-1)$ stages of shuffle-exchange I will be of the form $(I_{n-m+1}, \ldots, I_1, \tilde{I}_n, \ldots, \tilde{I}_{n-m+2})$. $\tilde{I}_i$ means either I_i or $\bar{I}_i$, complement of I_i. After a shuffle I will be of the form $(I_{n-m}, \ldots, I_1, \tilde{I}_n, \ldots, \tilde{I}_{n-m+2}, I_{n-m+1})$. So destination tags for the two inputs of a switch will not differ in any of the bits $O_{n-m+1}, \ldots, O_1$ iff none of the equations for $O_{n-m+1}, \ldots, O_1$ contain I_{n-m+1}. From lemma 3 we know that after an exchange operation at stage i, O_{n-i+1} is set to I_{n-i+1} or to $I_{n-i+1} + O_j$, $j < i$, whereupon O_j is set to $I_{n-i+1} + O_{n-i+1}$. So the equations for $O_n, \ldots, O_{n-m+2}$ are either independent of I_{n-m+1} or if they contain the term I_{n-m+1} then the equation for some O_j, $1 \leq j < (n-m+2)$ will also contain that term. ∎

Theorem 2 *$\mathcal{LC}$ permutations are routable using the algorithm 2 in π-network.*

Proof: From lemmas 2 and 3 it follows that after routing one stage of shuffle and exchange the resulting permutation is still an $\mathcal{LC}$ permutation but it could be different form the earlier one. So to distinguish this, we use superscript for the matrix $P_{n\times(n+1)}$. So the P matrix in the $\mathcal{LC}$ permutation for stage i is indicated as P^i. The P matrix for the first stage denoted as P^1 is same as the P matrix in the original $\mathcal{LC}$ permutation.

Consider an input $I = (I_n, \ldots, I_1)$ with destination $O = (O_n, \ldots, O_1)$. Let I be of the form $D = (D_n, \ldots, D_1)$ after routing first n stages using the algorithm. From lemma 3 and the discussion following the lemma, at any stage i, $1 \leq i \leq n$, D_{n-i+1} is same as O_{n-i+1} if the destinations have different $(n-i+1)$-bit, which is true only when $P^i_{i1} \neq 0$. Otherwise, $D_{n-i+1} = O_{n-i+1} + O_j$, $j < (n-i+1)$. Therefore we have,

$$D_{n-i+1} = O_{n-i+1} + \bar{P}^i_{i1}\, O_j,\ 1 \leq i \leq n \qquad (3)$$

Thus the equations for $D_n, \ldots, D_1$ will be of the form given below.

$$\begin{aligned} D_n &= O_n + \bar{P}^1_{11}\, O_j, \quad \text{for some } j < n \\ D_{n-1} &= O_{n-1} + \bar{P}^2_{21}\, O_j, \quad \text{for some } j < n-1 \\ &\vdots \\ D_1 &= O_1 \end{aligned}$$

We can rewrite these equations such that they are in the Ω characteristic equation form. For example we can rewrite the equation for D_n as given below.

$$O_n = D_n + \bar{P}^1_{11}\, O_j, \quad \text{for some } j < n$$

But, O_j can again be rewritten in the form given above. In general we can substitute $D_k + \bar{P}^k_{k1}\, O_j$, for some $j < k$, for O_k. Proceeding in this manner we obtain the equation for O_n only in terms of D's. In the same manner we can rearrange equations to get equations for all O's as given below.

$$O_i = D_i + F_i(D_{i-1}, \ldots, D_1),\ 1 \leq i \leq n \qquad (4)$$

Clearly these equations are in Ω form hence routable by

the last n stages of the π–network.

As an example consider the $\mathcal{LC}$ permutation in figure 5 characterized by the following set of equations.

$$O_3 = I_1;\ \ O_2 = I_3;\ \ O_1 = I_3 + I_2$$

Here O_3 does not contain I_3 but O_1 does. Hence substituting $I_3 + I_1(= LF_3) + I_2(= LF_1)$ *in all the occurrences of* I_3 and performing a shuffle on input bits we get the following set of equations which characterize $\mathcal{LC}$ permutation for the second stage.

$$O_3 = I_2;\ \ O_2 = I_3 + I_2 + I_1;\ \ O_1 = I_2 + I_1$$

One can verify that these equations hold after the first stage of switches. D_3 is given by the following equation.

$$D_3 = O_3 + O_1$$

In a similar manner we can obtain equations for D_2 and D_1 as given below.

$$D_2 = O_2;\ \ D_1 = O_1$$

Rewriting these equations we get,

$$O_3 = D_3 + D_1;\ \ O_2 = D_2;\ \ O_1 = D_1$$

which are in characteristic Ω *form, hence routable in the last three stages of the network (same as* 8 input/output Ω network).

3.3 $(2n-1)$–stage Shuffle Exchange Network

In the proof given above, we showed that $D_1 = O_1$. This implies that all the switches in the last stage are set straight. Hence, we can eliminate all the switches in the last stage. So, we need only $(2n-1)$ stages of shuffle exchange and a perfect shuffle. However, we can eliminate this shuffle at the output as follows.

We change the algorithm to treat destination tags as if a shuffle was performed on them. i.e., O_i is treated as $O_{(i+1) \bmod n}$. Let the given permutation be denoted as Π. With the modification the algorithm treats as if a shuffle was performed on Π. Hence in effect it routes $\Pi' = (\sigma\Pi)$. After routing for $(2n-1)$ stages a σ is required to route Π' correctly. So, after $(2n-1)$ stages we have routed $(\sigma^{-1}\Pi')$ correctly. But, $\sigma^{-1}\Pi' = \sigma^{-1}\sigma\Pi = \Pi$. Hence,

Theorem 3 *$\mathcal{LC}$ is routable in $(2n-1)$–stage shuffle exchange using the modified algorithm described above.*

4 Conclusions

In this paper we have presented algorithms to route $\mathcal{LC}$ permutations on Benes, π and $(2n-1)$–stage shuffle exchange networks. Since there will not be any conflicts in the first n stages of the Benes network if the permutation is in Ω^{-1}, this algorithm routes Ω^{-1} permutations as well. In fact the class of permutations routable using the algorithm given in this paper is much larger than $\mathcal{LC}$ class. With a similar argument any Ω permutation is routable using the algorithm in π network. It is interesting to note that it routes all permutations in $R(2)$ for 4 input/output Benes network. However this algorithm does not route all Ω permutations in Benes networks, with $N > 4$. If the permutation is known to be Ω then it can be routed by setting the first $(n-1)$ stages of the Benes network straight as suggested by Nassimi and Sahni [6].

References

[1] J. Beetem, M. Denneau, and D. Weingarten. The GF11 Supercomputer. In *Int'l Symp. on Comput. Arch.*, pages 108–115, 1985.

[2] V. E. Benes. On rearrangeable three–stage connecting networks. *The Bell System Technical Journal*, XLI(5):1481—1492, 1962.

[3] J. Lenfant. Parallel permutations of data: a Benes network control algorithm for frequently used permutations. *IEEE Trans. on Computers*, c–27(7), 1978.

[4] M. C. Pease, III. The indirect binary n–cube microprocessor array. *IEEE Trans. on Computers*, c–26(5), 1977.

[5] D. Nassimi and S. Sahni. Parallel permutation algorithms to set up the Benes permutation network. *IEEE Trans. on Computers*, c–31(2):148—154, 1982.

[6] D. Nassimi and S. Sahni. A self–routing Benes network and parallel permutation algorithms. *IEEE Trans. on Computers*, c–30(5), 1981.

[7] A. Waksman. A permutation network. *J. Assoc. Comput. for Mach.*, 15(1), 1968.

[8] P. Yew and D. H. Lawrie. An easily controlled network for frequently used permutations. *IEEE Trans. on Computers*, c–30(4), 1981.