計算機アーキテクチャ 111－4
（１９９５．３．１０）

並列コンピュータのための高速再配置可能合網

アイサム．A．ハミッド
情報環境系、情報デザイン学科
東北芸術工科大学
990山形市上桜田200番地

本論文は、ベンズ IN (Interconnection Network) の並列処理への応用が、より実際的となるような高速制御アルゴリズムについて考察している。まず、サイクリック･キューブ･トポロジーに基づいた CCE (Cyclic Cube Engine) と呼ぶ高速並列計算モデルを構成する。但し、$\Phi = N^{(1+1/h)}$ とする。ここで、Φ と N は、それぞれ、計算モデルを構成する処理要素の数と入力の数を表す。また、h は $2 \leqq h \leqq \log N$ を満たす任意の整数である。次に、このモデルを用いて、任意のパーミュテーションを $O(h \log_2 N)$ の時間で実現可能とする ベンズ IN の並列セッティングアルゴリズムを与える。これは、前述の並列セッティングアルゴリズムを *Accelerator* と呼ぶもう一つの高速アルゴリズムにより駆動することにより実現される。また、この手法による結果が、プロセッサ数が有限のもとで、一般的な nonshared モデルについて、任意のパーミュテーションに対して下限値内にあることが証明される。さらに、これらのアルゴリズムを用いて、任意のパーミュテーションに対して、ベンズ IN のスイッチを $O(h \log_2 N)$ の時間で設定するための高速並列アルゴリズムを構成している。

Highly Parallel Computation Model for Setting Rearrangeable Type Interconnection Network

ISSAM A. HAMID
Department of Information Design
Tohoku University of Art & Design
200 Kamisakurada, Yamagata city 990 Japan

Abstract:

We have presented here, for the Control Unit(CU) of Benes Interconnection Network(IN), a new fast parallel computational model named as; Cyclic Cube Engine (CCE), which depends on the Cyclic Cube topology, where the total number of processing elements is Φ such that $\Phi = N^{(1+1/h)}$, h is an arbitrary integer, such that $2 \leqq h \leqq \log N$. N is number of the data items consisting the permutation. We have presented on this model parallel algorithm for parallel settings of Benes IN in order to realize arbitrary permutation with a setting time of $O(h \log_2 N)$; (assuming is base 2) using $\Phi = N^{(1+1/h)}$processors (*i.e.*, $\Phi \geqq N$). This bound could be achieved by accelerating the parallel setting algorithm by function call of another very fast algorithm named as the accelerator. Using these algorithms we have constructed a fast parallel setting algorithm to set the switches of Benes IN for arbitrary permutation in parallel time of $O(h \log_2 N)$ where is $h < \log_2 N$. The parallel setting algorithm has been constructed depending on the CCE as the main computational structure suitable for setting Benes IN in parallel.

Index Terms: Complexity, Computational model, Graph theory, Rearrangeable interconnection network, Nonshared memory system, parallel algorithm, parallel computations, supersystems.

1. Introduction

Generally, there are two groups of models for the parallel architectures. The first group is concerned with the shared memory model. The second model is concerned with the many processors with local memories, each accessed exclusively by one processor. The second group should effectively be capable to permute data items among the Processing Elements; PEs by high permutability Interconnection Network(IN), which is the key feature for such system[Na79][Na80]. We have concentrated on the second group because, it is more practical to be constructed in comparison with the first group's model, which has many drawbacks such as; memory conflict and N−1 fanout per PE, (if in case N is the total number of data items or PEs). But, in the second group's model the IN becomes the most important part to synthesize powerful mappings. Typical Benes IN, consists of (2 $\log_2$N−1) stages. However for other type of INs such as Omega which has ((N/2) log N) SWs, and Gamma IN which has ((N/2) (log N+1)) SWs, it have almost the same or less number of SWs in comparison to Benes IN. But we should mention here that the mapping capability of Omega are much less than Benes IN, because Omega and Gamma or any type of IN of log N or (log N+1) stages, belong to blocking type INs, which are able to realize only specific group of permutations and cannot realize all N! permutations. But Benes IN is not easy to setup as the networks with $O(N^2)$ connections, (N is the number of inputs). For example, the looping algorithm [Op71][An77] sets sequentially, the SWs in the outer most stages; i.e., the first stage and the last stage, (as shown in Fig. 1), then sets the two upper and lower center-stage subnetworks in turn, (this is due to the recursive construction of Benes IN). As a result it runs in O(N $\log_2$N) time, which is worse than O(N) setup time of networks with $O(N^2)$ connections, (assuming N is base 2). Hence, there is a parallel algorithm[Na82][Na81] which sets the N-input in $O(\log^4 N)$ time. The main shortcoming of this algorithm as well as the others[Le81][Or87] is that it requires O(N) processors with the high fanout of $O(N^2)$ connections among them, because of using a shared memory computation model. Also, these previous algorithms are time consuming and not suitable for supersystems[Ha89]. Other researchers have also, approached the parallel realization to tackle this problem. Their work was either very general computation model for graph problems [Le81], or their algorithms[Na82][Ca87] have not sufficiently reduced the setting time to O(*h* $\log_2$N) to make this network efficiently approachable for parallel computations. Our construction is different from [Le81], because we do not depend on vertex coloring. Also, different from Nassimi[Na82], because we use very fast computational model with fast parallel permutation algorithm giving us a setting time complexity of O(*h* $\log_2$N).

Depending on such highly parallel permutation algorithm(*i.e.*, the accelerator) to permute N data items between Φ PEs, we have also, constructed a parallel setting algorithm to set the SWs of Benes IN to map arbitrary permutation.

In this paper, Sec.1, has given the introduction. Sec.2, has given the assumptions and notations. Sec.3, has discussed a new computation model for the CU of Benes IN named as Cyclic Cube Engine(CCE), which depends on the Cyclic Cube IN given by [Pr81]. Sec..4 has given a parallel permutation algorithm named as accelerator, depends on the radix-sort, implemented on the CCE. Sec.5, has discussed the realization of the parallel setting of the SWs' of Benes IN and how the accelerator has been used to set Benes' SWs for arbitrary permutation. Sec.6 has devoted for the conclusions.

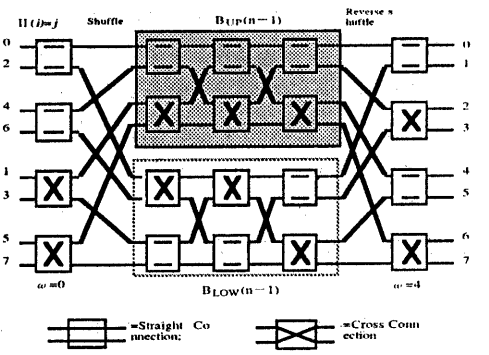

Fig. 1 . Benes Interconnection Network; (B(n=3)), for =8.

2. Assumptions and Notations

The following assumptions and useful characteristics are used throughout this paper:

(1) Each Processing Element; PE(*i*) has three registers, β (*i*), γ (*i*), μ (*i*), which correspond to that PE, and enough memory to hold one record; R(*i*), (this includes also, the field F(*i*)).

(2) The distinction between null and a record in a PE is possible, and one bit tag could be used.

(3) (:=) represents an assignment, (←) represents the connection and routing of two neighboring processors (has one unit route O(1)), and (N) represents an exchange connection and has O(1) if two way transmission is allowed.

(4) i_q, represents the bit q of the binary representation of *i*. Consequently, the PEs can be selected according to the mask specified by the bit q. When no mask is used all PEs are enabled, (instructions are executed only on enabled PEs.).

(5) $i_{\overline{q}}$, represents the complementary of bit q of the binary representation of *i*.

(6) $i(^{q}\ _{r})$ represents the selected bits, of the binary representation of *i* starting from bit q to bit r, i.e., $i(^{q}$

$_{s^n}$)=i_{s^0},...,i_{s^n}.

(7) The complexity of the algorithm has been measured in terms of the number of unit routes needed to execute it, assuming for simplicity that the arithmetic operations performed locally at each PE are executed in O(1) time.

(8) Every processing element, PE(i) in our evaluation represents a module, which contains an operand register, a field memory locations, and basic arithmetic and logical capabilities.

(9) The computation model is a nonshared memory model of an SIMD type (Single Instruction Multiple Data) machine, i.e., all instructions and data are loaded to a number of PEs which communicate between each other through a cyclic cube topology.

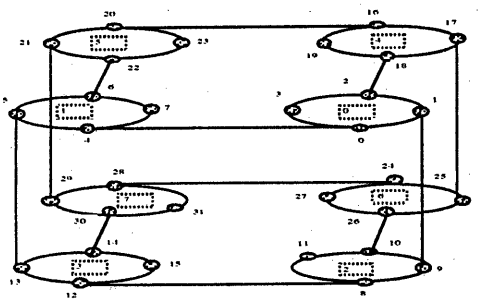

Fig. 2 . The structure of the Cyclic Cube Engine computational model.

3. The Cyclic Cube Engine (CCE)

The computation model used in our work is a nonshared memory model. All instructions and data are loaded to a specific number of processors which communicate through both cubic and cyclic configuration. The construction of CCE has been depended on the characteristics of cyclic cube computer(CCC) given previously by Preparata[Pr81]. The topological characteristics of CCE have been extracted from CCC[Pr81].

The CCE, is a network of identical processing elements, as shown in Fig. 2, where each *Processor Element (PE(i))* contains an operand register, a field memory locations, and basic arithmetic and logical capabilities.

Let N be the number of input or the element of a permutation, as base 2, i.e., N=2. R(i) represents a record located in PE(i), and F(i) represents the record's field. Let for some k such that, $2^k \geqq n$ and, $1 \leqq k \leqq n$. Then each *PE(i)* has n+k-bit address F(i) expressed as (x,y) of integers, where x has n bits length ($0 \leqq x < 2^n$), and y has k bits length ($0 \leqq y < 2^k$), such that; $x\ 2^k + y = R(i)$, where R(i) represents the address of PE(i), whereas, $0 \leqq R(i) < 2^{n+k}$, $N=2^n$, and the number of Φ processor elements; (PEs) consisting CCE is; $\Phi = 2^{n+k} = N^{(1+1/h)}$; where h =Nn/kN, (where N..N, represents the ceiling function,) such that, $2^k \geqq n$. Each PE has (n+k) bit address R(i) expressed as (x,y) of integers. Due to Fig. 2 each PE has *three* interconnection ports:

Next(x,y) is connected to *Before*(x, (y+1) mod 2^k), *Before*(x,y) is connected to *Next*(x, ($y-1$)mod 2^k), and *Cycle*(x,y) is connected to *Cycle*($x+Y\,2^y$, y), where $Y=(1-2\ x_y)$. (Where mod; represents the modulus calculation.). Recall that x_y is defined according to Sec.2, assumption 4.

The ports *Next* and *Before* are within the cycles. While the ports *Cycle* is between cycles which being connected as the cube configuration (Fig. 2).

It was mentioned in the appendix of [Pr81] that CCE can emulate Benes IN in time of O(log N), if in condition that the permutation is precomputed. However, this statement gives us the hint to choose CCE as the main computational model for the CU of Benes IN.

3.1. Partitioning CCE into 2^h-Group-Cycles

Let us observe in this section the binary representation for the addresses of the PEs which consisted the CCE. We have tabulated the addresses of the PEs of the CCE model by a two dimensional table (as shown in Fig. 3). Every row represents the PEs' addresses for a cycle. While columns represents the addresses of PE of 2^n PEs per columns for the n-Cube between n-Cycles, such that every column represents one dimension related to the cube configuration. For example, the first column (col 0) has PEs of addresses, (PE(0), PE(4), PE(8), PE(16)), which represents the horizontal connections between cycles of the CCE (See Fig. 2). Each individual PE at one column are different from the PE at the neighbor column by a Hammming distance of one because they are in one cycle. We can see from Fig. 2, and Fig. 3, that the second column represents the vertical connection between cycles. The third column represents the diagonal connection between cycles. While the forth column has no specific connection between cycles but within the cycles. This column works as a supporting element or auxiliary memory for records processed in the individual cycles.

We can observe also from the binary representation of every column as shown in Fig. 3, the following characteristics, which are useful for the algorithms derivation. The k low significant bits for all PEs of every column are equal while they are different in their n most significant bits. In contrast, all PEs of every row have equal n most significant bits and different k most significant bits. Due to these observations we can partition CCE into obstructed groups. Such an obstruction can be done according to the table's rows, i.e., the CCE's cycles. Therefore, due to this partitioning concept we define, the *2^h-group-cycles*, according to the following definition.

Definition 1

2^h-group-cycles, is as a sequence of PEs ($h=\lceil n/k \rceil$, $0 \leqq h \leqq n$) of 2^h consecutive rows or cycles. Each column of this group is a 2^h-groups of PEs. All addresses indices in *2^h-group-cycles* have the *same* h^{th} MSB (Most Significant Bit) of the h MSBs, and all other indices varies in only the other LSBs of the binary representation of PE. ■

For example, if h=2, then the 2nd MSB of the h=2 MSBs, is the fourth bit i.e., 2^h-bit=fourth bit, which is the dominated partitioning bit such that we can have two groups of rows. The first four rows has the fourth bit of 0, and the other four groups has the fourth bit of 1. In the same way, we can have four groups of 2^{h-1}-*group-cycles*, of which the bit, $(n+k-2)$=3rd MSB; is either 0 or 1. Such partitioning assists us to construct our algorithms, as will be shown later.

4. The Parallel Permutation Algorithm- (The Accelerator)

We discuss here, the parallel permutation algorithm depending on a parallel form of radix-sort algorithm[Pr78], and then implementing it using the CCE given in Sec.3. Please note that, in this paper the setting time for this type of network is proven to be within the lower bound which is shown to be proportional with the propagation delay of Benes IN itself, (i.e., $O(\log_2 N)$). However, here, we primarily concern with the development of efficient algorithms to sort and permute data on nonshared memory model, i.e., the cyclic cube engine, given in Sec.3. For the both problems (i.e., sorting and permutation), we assume that; there are $N=2^n$ records. Initially, record R(i) is in PE(i), $0 \leq i < N$. Each record has a field F(i). The field corresponds to one record into own processor' s memory.

We develop an algorithm to perform arbitrary permutation in $O(h \log N)$ time on the CCE when $\Phi=N^{(1+1/h)}$ PEs with a connection of degree *three* per processor (i.e., ε =3). In fact, the algorithm to determine switch settings is almost identical to the permutation algorithm. For the permutation problem, $F(i) \in [0, N-1]$, $0 \leq i < N$, and record i is to be relocated to PE(F(i)), $0 \leq i < N$. However, N records can be permuted on $O(1)$ time using the *Shared Memory Model*(SMM), and N PEs[Hi77] [Pr78]. PE(i) first writes its record into location F(i) of the common memory and then reads back the record in location i. However, the SMM is very difficult (if not impossible) to be implemented on VLSI, because N− 1 fanout connection per PEs is needed for construction[Pr78].

Also, the fact remains that no actual PEs array have been built that are based on the SMM because, it is not feasible to allow Φ processor to access Φ memory addresses, simultaneously. Therefore, the more realistic assumption is that each processor has its own private memory and PE can access data through an IN (e.g., CCE). In general, a parallel model is reasonable if the number of PEs each PE can communicate with is bounded by a constant[Go82] as is the case with the CCE. We have shown through the parallel algorithm that the CCE, is superior for the parallel permutation algorithm that is useful for the parallel setting algorithm for the CU of Benes IN.

Our algorithms in this section have a new nonshared computational model represented by the CCE. These fast algorithms require $O(h \log N)$ time on a model of $\Phi=(2^{n+k})$PEs which has a cyclic cube topology, and capable to realize arbitrary permutation, as will be explained here after. Please note that $\Phi \geq N$ (i.e., $\Phi=N^{(1+1/h)}$), therefore the number of Φ PEs are tolerable to N, i.e. , every PE may process only one record and, N should always match the Φ PEs, such that to not let more than one record applied to one processor.

4.1. The construction of the Parallel Permutation Algorithm

This section consists from three subsections. One represents an informal description with an example, the other gives the formal details for that construction, and the third, represents the analysis of these algorithms.

Our permutation algorithm is a parallel version of MSD (Most Significant Digit) radix sort. The radix=2^k. Then $\lceil n/k \rceil$ digits of F(i) are used. The following rules corresponds to how to obtain the MSD radix digits.

1) The binary representation of F(i) is obtained.
2) The k MSB yield the MSD (Most Significant Digit).
3) The next to the k bits give the next digit and so on.

For $0 \leq i < N$, $2^n=N$, R(i) represents record which is initially located in PE(i). F(i) represents a field in record R(i). (F(0),.....F(N− 1)) define a permutation of (0,.....N− 1). Recall that, F(i) is related to the permutation element i, and R(i) is related to a record in PE(i). The records are to be *permuted* so that following the permutation, R(i) is in PE(F(i)). This permutation is to be performed on the parallel computation model of $\Phi = N^{(1+1/h)}=2^{n+k}$ PEs, connected as cyclic cube topology where, $h=\lceil n/k \rceil$, $1 \leq k < n$, $n=\log_2 N$.

4.1.1. The Formal Description

The language construct for each j: (*cond j*) pardo (*action*) odpar, which has been used in the algorithm construction of this paper, indicates that all instructions (*action*) corresponding to values of j satisfying (*cond j*) can be performed simultaneously[Hi86]. However, it is Algol-like language. Please note that due to Sec.3, each PE of the CCE is basically counts up time at each time unit (numbered t) it tests a simple logical condition involving x,y, and t. Depending on this test either it does nothing or it exchanges operands or it exchanges operands and performs an operation on them. The algorithm performs a sequence of basic operations on a pairs of data that are successively, 2^{n+k-1}, 2^{n+k-2},...2^0=1 locations apart. The execution of an operation for all operands in a cycle of CCE requires 2^k time units (i.e., the length of the cycle in CCE). This computation can be pipelined (overlapped) with the analogous operation for any i, $k \leq i < n+k$.

The algorithm makes the records in each *group-cycles* to be leveled and gathered in parallel using CCE of $N^{(1+1/h)}$ PEs. The leveling of a record in a 2^h-*group-cycles* represents the number of the records preceding it in that group. Therefore, the procedure *LEVEL*, shown in Fig. 5, which works recursively determines this number for each record in every group of the *group-cycles*, in parallel. It

divides a 2^h-*group-cycles* into two 2^{h-1}-*group-cycles*. If L(*i*) is the level of a record (if any) in PE(*i*) and stored in its register β (*i*). Also, Σ (*i*) is the total number of records in the 2^{h-1}- *group-cycles* containing PE(*i*) and stored in its register γ (*i*). Then the level of a record in a 2^h-*group-cycles* is L(*i*) if i_{h-1}=0, (note that i_{h-1} =0 for the upper 2^{h-1} of a 2^h-*group-cycles*, in Fig.3 and Fig.4.), or L(*i*)+ Σ ($i_{I(h-1)}$), if i_{h-1}=1. (Where Σ (*i*) represents the total number of records in the 2^{h-1}-*group-cycles* including PE(*i*), and L(*i*) is the level of the record in the register β (*i*) of the PE(*i*), (if any) within the 2^{h-1}-*group-cycle*.) Then, unfolding the recursion yields the iterative procedure *LEVEL*. We can see that *LEVEL* uses O(*h*) PE time and exactly *h* unit routes. After leveling the records the procedure *GATHER* shown in Fig. 6, gathers the records within each 2^h-*group-cycles*, such that they are moved to consecutive PEs. Let L(*i*) be such that the record (if any) in PE(*i*) is be re-located to the R(*i*)th PE in the 2^h-*group-cycles*. *GATHER* is achieved by first re-locating all records in the *group-cycles* to PEs such that the PE indices and L(*i*) agree in bits 0 and 1; and so on until records have been routed to the correct PE. Fig. 5, shows the formal recursive construction of the algorithm named by the procedure *LEVEL* . Whereas, Fig. 6, shows the procedure *GATHER(h).*

The re-locations of records between PEs may cause that, some PEs' record be over written by the others, such that we may have what is called as *collision* in PE(*i*) whose records may be overwritten by incoming record from PE(i_{γ^j}). But, in our construction we do not have such an action. In order to prove this claim, let us consider for instance, the PE(*i1*) and PE(*i2*), which are in the same 2^h-*group-cycles*. Hence, leveling these records using the algorithm of Fig.5, we may have; $|L(i1)-L(i2)| \leqq |i1-i2|$, if the records of those PEs are overwritten by the record of; for instance, PE(*i3*) then; $i3=(i1(^{n+k-1}_{\gamma^{j+1}}), L(i1(^{\gamma^j}_{0}))) = (i2(^{n+k-1}_{\gamma^{j+1}})), L(i2(^{\gamma^j}_{0})))$, this implies, $2^{\gamma^{j+1}} > |i1-i2|$, and $|L(i1)-L(i2)| \geqq 2^{\gamma^{j+1}}$, therefore, $|L(i1)-L(i2)| > |i1-i2|$, which contradicts with the earlier inequality. Therefore our claim is satisfied.

The total realization of the parallel permutation is shown in Fig. 7, which shows the algorithm named as the procedure *PARALLEL-PERMUTATION*(n,*h*), that works to accelerate the parallel setting's algorithm of Benes IN given in Sec.3. Fig. 7, represents the total realization of the parallel algorithm performed in Nn/*h*N passes. The routing in each pass is done in parallel at each 2^h-*group-cycles*, note that, at each pass there is only one record per row. As shown in Fig. 7, at first the algorithm works to re-locate R(*i*), in the pass; for instance *p* to column σ (*i*), where σ (*i*) =F(i^{h-1}_{l})) such that, *h*=n−(*p*− 1)*k*, *and l*=max(*h*− *k*,0). Therefore, at first the record in each row is replicated over all PEs in that row, and then deleted except for the one in the proper row.

5. Parallel Setting Algorithm for Benes IN.

In this section we will represent the main parallel setting algorithms of Benes IN' s realization algorithms constructed on CCE model of Sec.3, and accelerated by parallel permutation algorithm given in Sec.4. It should be noted here, that there is a parallel setting algorithm for Benes IN[Na80][Na81], implemented on Mesh connected computer with a setting time of complexity of O((log^{4}N)), assuming N=Φ . This bound is because of using the Batcher sorting algorithm whose complexity is of O((log N)^2), to permute the data between Φ PEs.

5.1. Notations and Characteristics

We have presented here characteristics and realizations concerning with rearrangeable Benes IN; as following:

(1) Let Π (*i*)=*j* represents arbitrary permutation of N elements such that; *i*=(0,1,....N− 1), where 0≦ *i,j*<N, assuming N is base 2, i.e., N=2^n. (Note that, the small brackets; (....), represents an ordered set of elements.) Therefore Π (n) means the permutation as a function of n.

For instance, *i*=(0,1,2,3,4,5,6,7), *j*=(0,2,4,6,1,3,5,7), represents the perfect shuffle permutation.

Hence, for Π (0)=0, Π (1)=2, Π (2)=4, and so on. Also, let the inverse of this permutation be represented as V(Π (*i*)) = V(*j*)=*i*.

(2) According to the recursive structure of Benes IN[Wa68][Op71], Π (n) can be divided into two parts of subpermutations. Π_{UP}(n− 1) represents the upper subpermutation of 0≦ *i,j*<N/2, at the input of the upper middle stage, named as B_{UP}(n− 1), of Benes IN; B(n). Also, the Π_{LOW}(n− 1) represents the lower subpermutation of 0≦ *i,j*<N/2, at the input of the lower middle stage, named as B_{LOW}(n− 1) of the B(n) as shown in Fig.1.

(3) As has been presented by [Le81][Na82] we have also represented the required permutation; Π (n) by an undirected graph; G(Π (n)). We have represented here, every SW of B(n) IN as a vertex, while the edges represent connections between these vertices according to the required permutation. Consequently, we have two disjoint set of vertices. One represents the vertices corresponding to the input stage, i.e., first stage.

4) From the bipartite graph we can find subpermutation named as; *complete- subpermutation-group* which is defined by definition-2 below:

Definition 2

A *Complete-Subpermutation-Group (CSpG)*, represents a cyclic path from vertex $\lceil n/k \rceil$ with its edge connected to other disjoint vertex and so on till it return back to the same vertex $\lceil n/k \rceil$. ■

Knowing these *CSpGs*' we can find in parallel the switching states of the first and last stages, besides, the permutation of middle upper stages Π_{UP}(n− 1), and the permutation of middle lower stages Π_{LOW}(n− 1), of Benes IN, as will be shown later. Please note that the *CSpGs*' , represents sets of items which are a subpermutation groups of the original permutation.

(5) Let us define $\Delta(\omega, \varepsilon)$, as the function' s state of the SW; ε of the stage; ω. Hence, the parameter ω corresponds to the stage number, where $0 \leqq \omega < 2n-2$, and the parameter ε corresponds to the switch ε, where $0 \leqq \varepsilon < N/2$. $\Delta(\omega, \varepsilon)$ has only one of the two values, which represents either 0, or 1. The value 0 or 1 represents respectively, the setting of an SW to straight or cross.

(6) Let $H_{UP}(i/2)$ represents a subpermutation of $\Pi(i)$ which come from the output of the upper middle stage $B_{UP}(n-1)$ of B(n). For example for the perfect shuffle permutation, the $H_{UP}(i/2) = (0,3,4,7)$ which represents a subpermutation of the output permutation $\Pi(i)$, which come from the output of the $B_{UP}(n-1)$; see Fig. 1. It is interesting to note that $H_{UP}(i/2)$ has the following characteristics, which are related to B(n) construction: $0 \in H_{UP}(i/2)$, $|H_{UP}(i/2)| = N/2$, and if $i_1, i_2 \in H_{UP}(i/2)$, where $i_1 \neq i_2$, then $i_1 \neq (i_2)_{/\!/0}$, and $V(i_1) \neq (V(i_2))_{/\!/0}$, note that $V(i)$ represents $V(\Pi(i))$. However, the set $H_{UP}(i)$ defines a complete matching on the bipartite graph, $G(\Pi(i))$. In fact the set $H_{UP}(i/2)$, gives us enough useful information about the switch settings for stages 0 and $2n-2$, $\Pi_{UP}(n-1)$, and $\Pi_{LOW}(n-1)$ using the following rules: If $i \in H_{UP}(i/2)$ then; (1) $\Delta(2n-2, i/2) = i_0$, (2) $\Delta(0, V(i)/2) = (V(i))_0$, (3) $\Pi_{UP}(V(i)/2)) = i/2$, (4) $\Pi_{LOW}(N/2+(V(i)/2)) = N/2+(\Pi(V(i)_{/\!/0}))/2$. (Please note that all divisions are integer division, i.e., $\lceil n/k \rceil$.) These rules are related to the characteristics of the shuffle and reverse shuffle topologies' connection at the first and last stage of B(n) IN, respectively.

5.2. The Construction of The Parallel Algorithm for Benes IN

We have seen that the set $H_{UP}(i/2)$, are useful to find the Benes IN' s SWs' setting. In result, we would like to find a fast parallel method to find $H_{UP}(i/2)$.

A) The first step in the parallel algorithm, is to detect the *CSpG*, which is defined above as a cyclic path from and into again a certain vertex(Def.2). Therefore the procedure; *SUBPERMUTATION- GROUP* shown in Fig. 8, performs this task by finding the *subpermutation-groups*, which represent the components of the *CSpG*. Please note that, if C represents a certain *Subpermutation-Group(SpG);* resulted from the execution of procedure *SUBPERMUTATION-GROUP*, then C has the following characteristics: (1) If $i \in C$, then $i_{/\!/0} \in NC$. (2) If C_1, and C_2 are *SpGs'* , such that; $i \in C_1$, and $i_{/\!/0} \in C_2$, then $C_1 \cup C_2$ ($\cup$ denotes the union), represents a *CSpG* of the permutation graph; $G(\Pi(i))$. (3) $|C_1| = |C_2|$ if $i \in C_1$, then $i_{/\!/0} \in C_2$ and $i \in C_2$, then $i_{/\!/0} \in C_1$. Note that this algorithm works on the computational model presented in Sec. 3, where PEs in a 2^h-*group-cycles*, are linked together using a field F(*i*), (recall F(*i*) is the link field in PE(*i*)). As has been presented in Sec.4.2, an arbitrary permutation is, initially distributed over Φ PEs, such that every record is in one cycle of the CCE model. Therefore, the record *j* is in PE(*i*) of the cycle; *i*, and $0 \leqq i < N$ of the CCE. If $j \in \Pi(i)$ then for some $j' \in \Pi(i)$ such that; $j' = \Pi((V(j))_{/\!/0})$, note that $V(\Pi(i)) = V(j) = i$. Therefore, applying the algorithm of Fig. 8, we have for instance, *j* and *j'* be in different *SpGs'*, and *j* and $(j')_{/\!/0}$ are in the same *SpGs'*.

The time complexity of *SUBPERMUTATION-GROUP* algorithm is of O(*k*), which represents the parallel computation time within the cycles of 2^k PEs of the CCE. This algorithm is constructed such that it uses the direct connection of O(1) communication time, for data transfer between the PEs of CCE. Moreover, the loop operations between the Φ PEs in the cycles of the CCE, take a time of O(*k*), where 2^k represents the number of PEs in a cycle; (Sec.2). Fig. 9 give an example of how this algorithm works for instance, for the perfect shuffle permutation whose the *SpGs'* determined by the above procedure are; (0,3), (4,7), (1,2), (5,6).

(B) The second step is to find $H_{UP}(i/2)$ and $H_{LOW}(i/2)$ from the *SpGs'*, which are determined by the procedure *SUBPERMUTATION-GROUP*, in parallel. Here, these groups are localized to 2^h-*group-cycles*, (defined in Sec.3.1). Therefore, the procedure *DIMINUTION*(*h*)'s algorithm shown in Fig. 10 , diminish the permutation $\Pi(i)$ represented by the *SpGs'*, such that to determine D(*i*); which represents the minimum integer in the *subpermutation-group* (containing *i*, $0 \leqq i < N$). This procedure takes in use the direct connection of CCE. Hence, after for instance the iteration ϕ of the for loop, the register β which contains the field F(*i*) points to the register β of other PE at distance $2^{\phi+1}$, (please note that, the distance is measured along the *subpermutation-group*, mapped on the cycles of CCE(i.e., the 2^h-*group-cycles*)). Therefore, completing all the iterations till $\phi = h-1$, we may have D(*i*), be as the minimum integer of the *CSpG* that also contain *i*. Using the same example of the perfect shuffle permutation, the set of *subpermutation-group*, will be mapped onto 2^{h-1}-*group-cycles*, of CCE, such that every *subpermutation-group* is on one 2^{h-1}-*group-cycles*. Due to Sec.4, we have (for our example) four groups of *group-cycles*, such that every *subpermutation-group* is mapped on one *group-cycle*. Operate all groups of the *group-cycle*, in parallel to determine $H_{UP}(i/2) = (0,3,4,7)$. It is obvious that execution time is proportional to the partitioning time of the *group-cycle, i.e., O(h)*.

(C) This is the complete algorithm named as the procedure *PARALLEL-MAP-SETTING*(N,*h*,*k*), which calls the procedures presented above as A and B, as shown in Fig. 11. At each iteration the setting of the first and last stages of all B(*h*) Benes subnetworks of B(n) is determined in parallel. Let us apply the procedure *PARALLEL-MAP-SETTING* of Fig. 11 to set Benes IN shown in Fig. 1, for the perfect shuffle permutation. As well as N=8, and N=3, therefore for *h*=3, and the setting of the first stage; $\omega = 0$, such that, $\Delta(\omega = 0, Ni/2N) = \Delta(0,0) = 0$, $\Delta(0,1) = 0$, $\Delta(0,2) = 1$, $\Delta(0,3) = 1$. Also, $\Delta(\omega = 2n-2, Ni/2N) = \Delta(4,0) = 0$, $\Delta(4,1) = 0$, $\Delta(4,2) = 0$, $\Delta(4,3) = 1$. In addition, $\Pi_{UP}(Ni/2N) = (0,2,1,3)$,

$\Pi_{LOW}(Ni/2N)$=(5,7,4,6), These results are computed in parallel using the CCE' s cycles. From the sub permutations $\Pi_{UP}(Ni/2N)$ and $\Pi_{LOW}(Ni/2N)$ of the middle stages, we may have the permutation;

$\Pi_{UP}(\lceil i/2 \rceil)+\Pi_{LOW}(\lceil i/2 \rceil)$ =(0,2,1,3,5,7,4,6) as the input to the Benes IN of B(n−1)=B(2), i.e., h=2. Repeat the same procedure for h=2, we may have the following *SpGs'*, (0,3), (1,2), (4,7), (5,6), determined by the procedure *SUBPERMUTATION-GROUP* (as shown in Fig. 9). Also, by the procedure *DIMINUTION*(h), we may have $H_{UP}(\lceil i/2 \rceil)$=(0,3,4,7). Also, in the same way, $\Delta(\omega=1, \lceil i/2 \rceil)=\Delta(1,0)=0$, $\Delta(1,1)=1$, $\Delta(1,2)=1$, $\Delta(1,3)=0$. Also, $\Delta(3,0)=0$, $\Delta(3,1)=1$, $\Delta(3,2)=0$, $\Delta(3,3)=1$. Moreover, $\Pi_{UP}(\lceil n/k \rceil)$=(0,1,3,2), $\Pi_{LOW}(\lceil n/k \rceil)$= (5,4,6,7). Therefore the permutation which is on the input of Benes IN of B(1) is at the stage; ω =2, which is only one stage. Therefore, $\Delta(\omega=2, \lceil i/2 \rceil)=\Delta(2,0)=0$, $\Delta(2,1)=1$, $\Delta(2,2)=1$, $\Delta(2,3)=0$. Fig. 1, shows these settings for the perfect shuffle permutation on the Benes IN. Note that the rules (3) and, (4) are to be interpreted as being carried out on each 2^h-*group-cycles* or equivalently for each B(h) networks of Benes IN.

Please note that the statement denoted on Fig. 11, represents how to achieve the rule (3) and (4) of Sec.5.1. Whereas a division by 2 requires us to shift bits $h-1$,...1 one position right and defines the new bit ($h-1$) to be zero. Also, adding 2^{h-1} requires changing bit $h-1$ to 1. Therefore, the statements; 3 and 4 on Fig. 11, implement the rules (3) and (4) for each B(h) network. If the least element in a *CSpG*, is even then all elements in that class must be routed through the upper B($h-1$) network. The statements c is executed on PEs whose register $\beta(i)$ is zero. In this statement the SW's settings for the first stage of all B(h) networks are determined by the rule (2). When h=1, then B(h) networks have only one stage, and then the statement d terminates the operation. But when $h \neq 1$, the SWs settings for the last stage of all B(h) networks are determined by the statement e.

```
procedure LEVEL(h.k.n);
comment   make the records for each group of PEs. Recall that. β(i), γ(i), and μ
          (i) represent the registers of PE(i). such that. L(i) in register β(i), Σ(i) i
          n γ(i);
global integer array F(i). L(i), Σ(i);
β(L(i)):=0;
if F(i)≠null then γ(Σ(i))=1 else γ(Σ(i))=0;
do foreach ϕ:=h to n+k−h;
begin
  pardo μ(i//ϕ)←γ(Σ(i));
if i//ϕ=1 then β(L(i)):=β(L(i))+μ(i);
γ(Σ(i)):=γ(Σ(i))+μ(i);
  odpar;
end;
end LEVEL;
```

Fig. 5 . The algorithm for the procedure *LEVEL*(n.*k*. *h*).

```
Procedure GATHER(h,k,n)
comment   According to L(i) determined by procedure LEVEL. relocate the record
          s R(i) to the PEs;
global integer array R(i), L(i);
comment   F(i) is the field of the record R(i);
do foreach ϕ=h to n+k−h
  if F(i)≠null, and β(L(i))≠i_ϕ then
    begin
      pardo (R(i//ϕ)), β(L(i//ϕ)))←(R(i),L(i));
      odpar,
    end;
end GATHER;
```

Fig. 6 , The algorithm for the procedure *GATHER*(n,*k*,*h*).

```
procedure PARALLEL-PERMUTATION(n.k.h);
comment   On CCE of 2n+k PEs. the parallel permutation of 2n records. according to
          the field F(i) of the record R(i). can be done;
global integer array R(i). L(i);
begin
  if i(k 0)≠0 then F(i):=null;
comment initialize the remaining columns;
  h:=n;
comment the number of column in a 2h-group-cycles. is 2n;
    for p=1 until n/k :                                                  ↑
comment p is a pass of n/k passes of radix sort algorithm;               |
      pardo for each ϕ=0 to k;                          ↑                |
comment copy records over columns;                      | a              |
        pardo if F(i)≠null then R(i//ϕ)←R(i);           |                |
        odpar;                                          ↓                |
      odpar;                                                             |
  l:=max(h−k,0);                                  ↑                      |
comment bits. h−1,.....,l form the digit;         | b                    | d
  if i/2n ≠F(i(h−1 l)) then                       ↓                      |
  F(i):=null else F(i):=F(i);                                            |
  call LEVEL(h.k.n);                                                     |
  β(L(i)):=β(L(i))+ i/2n ·2l                                             |
  call GATHER(h.k.n);                                                    |
  g:=h; h=l;                                                             |
comment Each partition is 2l-group-cycles columns;                       |
end                                                                      ↓
comment relocate records to the first column;
begin
  pardo for each ϕ=0 to g−1;                   ↑
    if F(i)≠null then R(i//ϕ)←R(i);            | c
  odpar;                                       ↓
  end;
end
end PARALLEL-PERMUTATION
```

Fig. 7 . The algorithm for the procedure *PARALLEL- PERMUTATION*(n.*k*.*h*).

```
procedure SUBPERMUTATION-GROUP:
comment   find the subpermutations which represent the subpermutation-group;
global integer array F(i). Π(i). V(i);
β(F(i)):=(Π(i))//0;
comment   assign the register β(i) which contains the field F(i) with the value (Π(i)
          )//0;
β(F(i//0)) ← β(F(i));
call PARALLEL-PERMUTATION(n.h.k)
<V(Π(i)). β(F(Π(i)))> ← <i. β(F(i))>;
end SUBPERMUTATION-GROUP
```

Fig. 8 . The algorithm for the procedure *SUBPERMUTATION-GROUP*.

6. Conclusions

The construction of the control unit of Benes interconnection network, appears to be highly sequential in nature, through the controlling algorithms presented by

[Op71][An77][Na79][Ca87], but it can be parallelized using efficient model like the CCE model presented in this paper. This bound could be achieved by using an accelerator which can realize the parallel permutation problem in time of $O(h\log_2 N)$, on the CCE parallel processor model of $\Phi=N^{(1+1/h)}$ PEs (i.e., $\Phi>N$), where h is arbitrary, and $2 \leqq h \leqq \log N$. In [Ha89] we have examined the lower bound when $\Phi \leqq N$. We have given here, a fast algorithm for the same problem when $\Phi=N^{(1+1/h)}$, such that each item is assigned per PE. That algorithm has a bound of $\Theta(h \log N)$. The strength of such an algorithms comes from its ability to efficiently permute data sending them to the location where the next processing step will occur. Therefore, according to these algorithms we have constructed parallel setting algorithms which can set the switches of Benes IN in parallel time of $O(h \log N)$, in order to realize arbitrary permutation. This bound for setting Benes IN is a new bound that can make such network more practical for parallel computers.

```
procedure DIMINUTION(h);
comment this procedure diminute the permutation Π(h) to find out the minimum ele
        ment in every subpermutation;
global interger array F(h), D(i); integer array μ(i), γ(h);
begin;
  D(i):=i;
  parado foreach φ=0 to h−1
comment works in parallel on all 2^h−1 group-cycles
  γ(j_φ)←β(F(i))_{j_φ};
comment this assign F(h)=j during iteration φ=0, F(j_φ)=i_φ, therefore γ(F(j_φ))
        =j_φ;
call PARALLEL-PERMUTATION(h)
    (μ(γ(i)), β(F(γ(i)))) ← (D(i), β(F(i)));
  D(i):=min(D(i), μ(i));
  odpar;
end DIMINUTION
```

Fig. 10 . The algorithm for the procedure *DIMINUTION(h)*.

```
procedure PARALLEL-MAP-SETTING(n,h,k);
comment parallel algorithm for setting Benes IN;
global integer array  Π(0:  −1), D(0:  −1), V(0:  −1),
                      Δ(0:2n−2, 0:(  /2)−1);
begin
REPEAT: for h:=n  to 1  step −1
  begin
    ω:=n−h;
call SUBPERMUTATION(h);
call DIMINUTION(h);
comment for permutation Π(h)=j if j∈Π_{i*}(  i/2  ), then D(i):=0;
  D(i):=D(i_ω);
call PARALLEL-PERMUTATION(h);
comment this means β(V(i))←D(i):
    (β(V(i)),D(i)) ← (V(i),D(i));
comment setting of the first stage:                                  ↕ c
  if β(i)=0 then ω=i_ω and Δ(ω,  i/2  )=i, else Δ(ω,  i/2  ):=Δ(∞,∞);
  if h=1 then exit;                                                  ↕ d
comment assign the last stage:
  if D(i)=0 then γ(Δ(2n−2−ω,  i/2  ))←i_ω                            ↕ e
ccomment Updates Π(i) such that it corresponds to the Π_{i*}(h−1) if it is even;
  if β(i)≠b then μ(Π(i/2^ω))←μ(Π(i));
  μ(Π)(i_{h−1}.....i_ω i_{ω−1}....i_1)←μ(Π(i));                      ↕ a
comment routes the Π(i) to PEs whose index corresponds to the rules ], and
        :
  Π(i):=i^{(n−1}_{h−1}(D(i^{(h−1}_{1})));                            ↕ b
  end
goto REPEAT;
end PARALLEL-MAP-SETTING
```

Fig. 11 . The algorithm for the procedure *PARALLEL-MAP-SETTING(h, k,n)*.

REFERENCES:

[An77] Andresen, S., "The looping algorithm extended to base 2^t rearreangeable switching networks," *IEEE Trans. Commu.* vol. Com-225, pp. 1057-1063, Oct. 1977.

[Ca87] Carpinelli, J.D., Oruc, A.Y., "Parallel set-up algorithms for Clos networks using a tree-connected computer," *Second Int. Conf. on Supercomputing*, pp.321-327, vol.1, May, 1987.

[Kn72] Knuth, D.E., The art of computer programming-Volume 1/ Fundamental Algorithms, Addison-Wesley Publishing company, 1972, (pp.104-108).

[Ha89] Hamid, I.A., et.al., "A new fast parallel computation model for setting Benes Rearrangeable interconnection network," *Trans., IEICE*, vol.E72, No.4, April, 1989, pp.393-403.

[Hi86] Hillis, D., The connection Machine, MIT press, Camdridge, Massachusetts, 1986.

[Le81] Lev, G.F., et. al., "A fast parallel algorithm for routing in permutation networks," *IEEE Trans. Comput.* vol. C-30, pp. 93-100, Feb., 1981.

[Na82] Nassimi,D., et.al., "Parallel algorithms to setup the Benes permutation network," *IEEE Trans. Comput.* vol. C-31, pp. 148-154, Feb., 1982.

[Na81] Nassimi, D. et. al., " A self-routing Benes network and parallel permutation algorithms," *IEEE Trans. Comput.* vol C-30, pp. 332-340, May 1981.

[Na80] Nassimi, D. et. al., "An optimal routing algorithm for mesh connected parallel computers," *JACM, vol. 27.*, No.1, pp.6-29. 1980.

[Na79] Nassimi, D. et. al., "Bitonic sort on a mesh connected parallel computer," *IEEE Trans. Comput.* vol. C-27, pp. 2-7, Jan. 1979.

[Op71] Opferman, D.C., et.al., "On a class of rearrangeable switching networks," *Bell System Tech. J.* vol. 50, pp. 1579-1600, May-June 1971.

[Or87] Oruc, A.Y., et.al., "Programming cellular permutation networks through decomposition of symmetric groups," *IEEE Trans. Comput.* vol. C-36, No.7, pp. 802-809, July, 1987.

[Pr78] Preparata, F.P., "New parallel-sorting schemes," *IEEE Trans. Comput.* C-27, pp. 669-673, July, 1978.

[Pr81] Preparata, F.P., Vuillemin, J., "The cube-connected cycles: A versatile network for parallel computation," *CACM*, pp. 300-309, May, 1981.