PAPER NO.598

SUBCUBE MIGRATION IN MULTISTAGE INTERCONNECTION NETWORKS (PART-I)

BY

Muhammad Naeem Ayyaz

SUBCUBE COMPACTION IN MULTISTAGE INTERCONNECTION NETWORKS (PART-I)

Muhammad Naeem Ayyaz[1]

ABSTRACT

Multistage Interconnection Networks (MIN) can dramatically reduce architectural cost for high performance parallel processing. It is now possible to cost-effectively build small-scale to large-scale MIN-based multiprocessors. Most efficient use of such systems would be to make them available in multi-user and multitasking environments. Efficient schemes for resource management, *i.e.* subcube compaction and subcube migration, have been designed and implemented in this research for MIN-based systems. The schemes are *near-optimal* as well as *scalable* over the size of the interconnection networks.

Part-I of this study deals with subcube compaction. Some common and fundamental concepts related to both subcube compaction and subcube migration are developed in this part of the study. Part-II, [mna2], which is based on the concepts developed in part-I, deals with subcube migration.

Keywords

Multistage interconnection networks, multi-user multiprocessors, network partitioning, network reconfiguration, subcube allocation, subcube migration.

INTRODUCTION

Multistage Interconnection Networks (MINs) are suited for high performance parallel processing systems due to their versatile data permuting capabilities to support SIMD parallelism, and high throughput to support MIMD parallelism. The multiprocessors that we are concerned with here are based on a class of MINs called *fully partitionable* MINs [js87].

Because of the current advances in VLSI, it is now possible to build small-scale ($N < 32$), medium-scale ($64 <= N <= 128$) and large-scale ($128 < N$) MIN-based multiprocessors. Since, the potential applications running on these systems may not always require all the computing power at one time, the most efficient use of the system resources would be to make them available in a multi-user, multitasking environment. Efficient management of system resources in such an environment has been recognized as a major problem for ensuring high system throughput and utilization.

When a job request arrives in a multi-user multiprocessor environment, a complete k-dimensional subcube, or a k-cube, consisting of 2^k processors is allocated to a task that requires p processors, where $2^k \square p < 2^{k+1}$. In the case of MIN-based multiprocessors, this involves decomposing the multiprocessor into subsystems of different sizes by way of partitioning the underlying network into independent subnetworks [hj89] [js87]. During normal operation of the

[1] Department of Electrical Engineering, Univ. of Engg. & Tech. Lahore, Pakistan.

multiprocessor some subcubes may be busy in executing jobs, some may be idle. As such, when repeated allocation and deallocation of subcubes takes place, the current free subcubes may not always represent the largest subcubes that can be *formed* out of the current idle processors.

To ensure an efficient management of system resources, two resource management facilities can be provided. Firstly, the idle processors can be mutually combined, to form the largest possible, and as many as possible, free subcubes; this will be called *Subcube Compaction*. Secondly, tasks can be migrated, from time to time, to create larger free subcubes; this is called *Subcube Migration*.

In [j87] and [js87], Jeng and Siegel address a *subproblem* of the subcube compaction problem. They combine idle resources (repeatedly) at the subsystem level, until *one largest free subcube* is formed. To actually create that largest subcube, the system must be re-configured by way of *re-partitioning* the underlying network.

In this research, we give a new solution for subcube compaction problem. We devise a scheme to create rather the largest possible and as many as possible (not just one largest) free subcubes from the currently idle processors. The solution is also scalable and near-optimal over the size of the small-scale, medium-scale, and large-scale systems. The other study, [mna2], gives a near-optimal and scalable solution for the subcube migration problem.

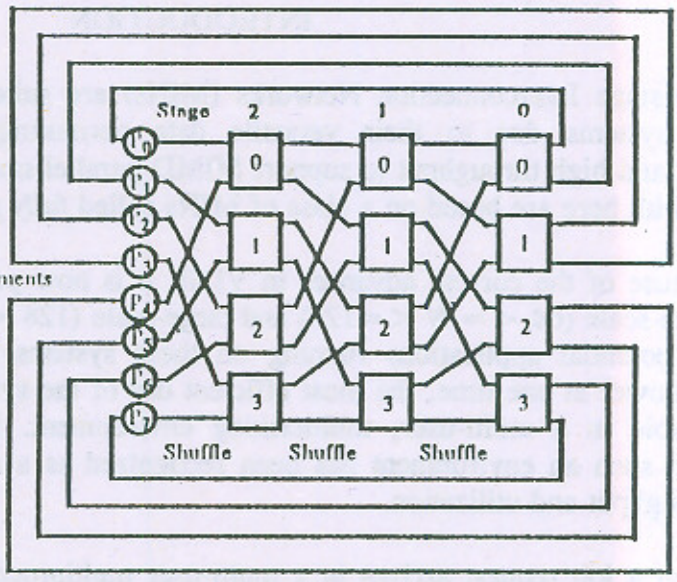


Figure 1. An MP_3, PE-to-PE configuration

SYSTEM CONFIGURATION OF MIN-BASED MULTIPROCESSORS

MIN-based multiprocessors have two basic system configurations [hj89]. One is the 'PE-to-PE configuration', in which the network is used to connect N processors on the input side to the same N processors on the output side, in a wrap-around fashion. The other configuration is the 'processor-to-memory configuration', in which N processors on the input side are connected, through the network, to N memory modules on the output side. However, only the 'PE-to-PE configuration' will be used here to study this problem.

We consider here a multiprocessor, named MP_n, whose N PEs are connected to each other through an n-stage, $n = \log_2 N$, omega network. Omega network is the standard model for a fully partitionable MIN. An MP_3, for example, is shown in Fig. 1. The input/output ports of the network, or the individual PEs to be connected with these ports to form a MP_n, are numbered from 0 to N - 1. The stages are labeled in a sequence n-1, n-2,..., 0, in left to right order.

PARTITIONING MP_N INTO DISJOINT SUBCUBES

Consider below recursive partitioning of an underlying omega network into two or more independent subnetworks of 2^k sizes, $0 <= k <= n$-1.

The *physical* address of a subnetwork/subcube will be denoted using a ternary symbol set, using characters {0, 1, X}. A 2-cube formed by PEs 10001, 10011, 11001, and 11011, for example, would be denoted by 1X0X1. If all switches of some stage i, $0 <= i <= n$-1, are set to the straight connection, this will partition the network into two $N/2$ size subnetworks. The first *low-order*
subnetwork has ternary address $X_{n-1} \ldots X_{i+1}\, 0\, X_{i-1} \ldots X_0$. The other *high-order* subnetwork has ternary address $X_{n-1} \ldots X_{i+1}\, 1\, X_{i-1} \ldots X_0$. $X_{n-1} \ldots X_{i+1}\, 0\, X_{i-1} \ldots X_0$ can be similarly recursively partitioned into another two $N/4$ size subnetworks by setting any arbitrary substage j, j not equal to i, to a straight connection, and so forth.

Such recursive decomposition of the MP_n can be represented with the help of a Dynamic 2-Tree, or, in short, a D2Tree, data structure. Its root node represents the n-cube MP_n. Each *leaf node* at level n-k is associated with either a free or a busy k-cube. The status of a free (busy) leaf node will be indicated on D2Tree by an 'F' ('B'). Each *nonleaf node*, say node I, at level l, $0 < l < n$, corresponds to a subcube that was further decomposed into another two (leaf and/or nonleaf) subcubes of dimension n-l-1 at level l+1. These two subcubes are created by straight setting the stage whose number appear inside the 'circle' of the parent node I, in the D2Tree figures.

SOME NOTATIONS

Let $S_{F,n}$ denote the set of all current free subcubes in the system, and $P_{F,n}$ the set of all free PEs contained in $S_{F,n}$. Let a set called the *ultimate set* $U_{F,n}$ contain a free i-cube for every ith '1' bit in the binary representation of $M = |P_{F,n}|$. This is that set of free subcubes that would form because of subcube migration operation. This is set is always guaranteed by the migration operation. Secondly, let a set called the *maximal set* $M_{F,n}$ denote the largest possible, and as many as possible, free subcubes formed out of $P_{F,n}$ as a result of the subcube compaction operation. In general, $M_{F,n} \square U_{F,n}$, as far as the size of the constituent subcubes is concerned.

SUBCUBE COMPACTION VIA RE-PARTITIONING OF MIN

If $S_{F,n}$ at some point contains more than one i-cube, for any i, the free PEs in $P_{F,n}$ can be combined with one another to form a maximal set $M_{F,n}$.

Example 1: Consider an example that we picked from actual simulation. In Fig. 2 (a), $S_{B,6}$ = {00101X, 10011X, 101011, 101010}, and $S_{F,6}$ = {0X0XXX, 11XXXX, 0X11XX, 0110XX, 1011XX, 1000XX, 10100X, 10010X, 00100X}. Re-partitioning of the network would produce an
$M_{F,n}$ = {XXXX0X, X1XX1X, X0001X, X0111X, 00011X} as shown in Fig. 2 (b), with four less free subcubes than $S_{F,6}$.

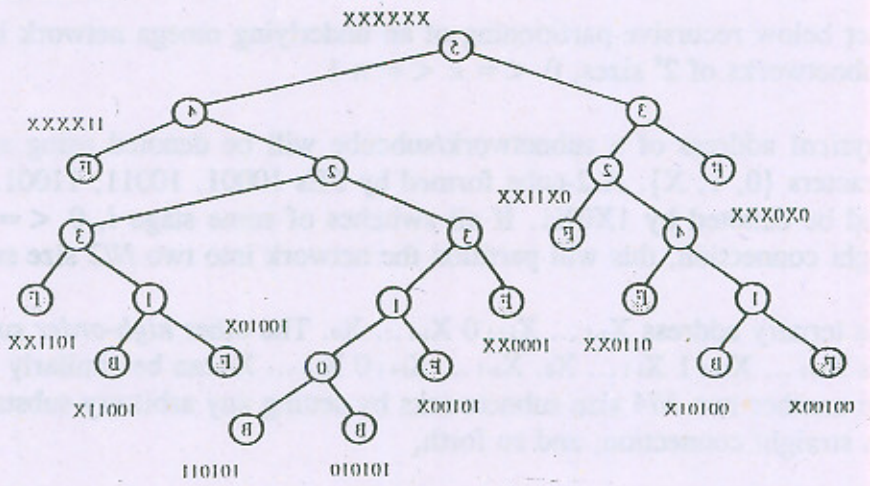


Figure 2 (a), Original D2Tree for Example 1

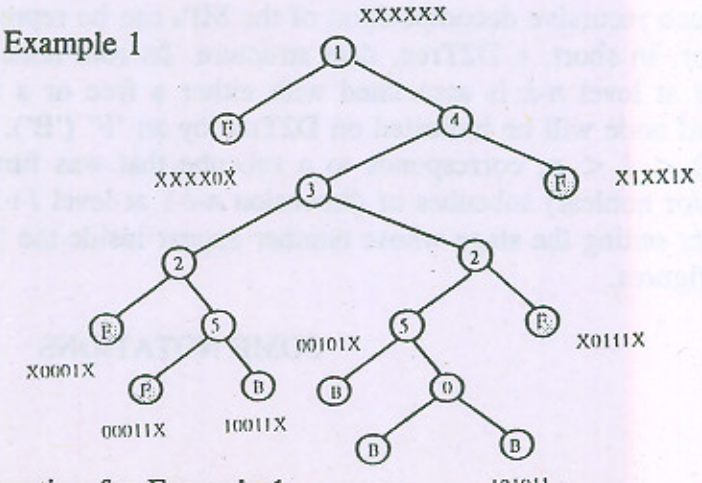


Figure 2 (b), Illustrating subcube Compaction for Example 1

DISJOINT VS PARTITIONALLY-DISJOINT SUBCUBES

During routine operation of the multiprocessor, or, for that matter, before application of the compaction operation, all free and busy subcubes of the multiprocessor are not only mutually disjoint, but also they are in fact *Partitionally-Disjoint* (PD) also. What this means is that for any given PD set of subcubes, which make up a MP_n, it would be always possible to fully partition the multiprocessor, in one or more possible ways.

Example 2: The set of mutually disjoint subcubes {10XX1X, 10XX0X, 001XXX, 0X00XX, X11XXX, 110XXX} forms an MP_6. The set is however not PD, because there does not exist any bit position at which the system can be partitioned, even into two subsystems.

OPTIMAL AND NEAR-OPTIMAL SOLUTIONS FOR COMPACTION

If we (a) focus on the free PEs in $P_{F,n}$ and, somehow, combine or merge them with one another so as to form a set of largest possible free subcube(s) of $M_{F,n}$; and, in due course, (b) *do not* break up any of the currently busy subcubes also; and, simultaneously, (c) make sure that the final set ($M_{F,n}$ U $S_{B,n}$) (U here means set union) proves PD too, then the compaction would result in a multiprocessor in which the idle resources are optimally compacted. Achieving an optimal solution for subcube compaction is a *NP*-hard problem. However, the real-time compaction scheme (scaled and tuned specifically for small-scale, medium-scale and large-scale systems) presented later is so powerful and efficient that, in most situations, the finally achieved solution will be at least *near-optimal*. In that case, we will be actually creating a *near-maximal* set, denoted as $NM_{F,n}$ from the *M* free PEs. Formal definition of a near-optimal solution will be given later in section 10.

A SCALABLE SOLUTION SPACE

To pursue an optimal or near-optimal solution for compaction, we need to have in hand a solution-space in which all different free PEs can mutually combine and form the largest possible subcubes of the maximal set $M_{F,n}$ or the near-maximal set $NM_{F,n}$.

W. Lin [lw86] and Siegel [js87] proclaim that, in situations like this, the free subcubes must first be disjoined into the *minimal subcubes*, or *minterms*, that represent individual PEs. Such a solution space, however, would result in unacceptably high space and time complexity for the solution. Keeping this in view, we devise a scheme whereby we do not at all need to disjoin $S_{F,n}$ into individual PEs. For small-scale systems, it will be equally good to disjoin $S_{F,n}$ into only *an intermediate set of minterms*, denoted by $S_{F,min}$. In this case, $|S_{F,n}| <= |S_{F,min}|$

$<= |P_{F,n}|$. Employing this disjoining scheme will considerably reduce the size of the solution space, and, hence, the space and time complexity of the solution.

Example 3: If an $S_{F,5}$ = {00XXX, 11XXX, 100X1}, then disjoining it into 17 individual PEs would produce the same largest free subcubes in the solution space as disjoining it into only 7 minimal subcubes of the set $S_{F,min}$ = {001XX, 111XX, 000X0, 000X1, 110X0, 110X1, 100X1}. Similarly, if $S_{F,5}$ = {10XXX , 001XX, 111XX}, it is equally good to disjoin it into only four minimal subcubes as in $S_{F,min}$ = {111XX, 100XX, 101XX, 001XX}.

For large-scale systems, $S_{F,n}$ will not be disjoined at all and will be used as such in building the solution space

ACTUAL GENERATION OF THE SOLUTION SPACE

The above considerations lead us to create the desired solution space in the following manner.

Let a represent dimension of the smallest subcube in $S_{F,min}$ and/or $S_{B,n}$. The solution-space will be represented in the form of a graph $G < V, E >$. G in turn would consist of $(m-a)+1$ subgraphs $G_i <V_i, E_i>$, $a <= i <= m$, where $m = floor\ (\log_2 M)$. Here V_i represents the set of nodes of a subgraph G_i wherein each distinct node would be representing a different possible i-cube of the solution-space. E_i would represent *inter-subgraph* edges, i.e., edges from the nodes of subgraph G_{i+1} to the nodes of subgraph G_i, $a <= i < m$. The solution space is constructed as follows:

Step 1: In accordance with the number of 1s in the binary representation of each i-cube, $a <= i <= n-1$, of the sets $S_{F,n}$ and $S_{B,n}$, separate the i-cubes into groups with the same number of 1s. Now represent each such i-cube by a distinct node on the respective subgraph G_i.

Step 2: Compare each a-cube in one group, in G_a, with every a-cube in the successive group on its right. If the two a-cubes are the same in every position except one, then combine them and insert the resulting $(a+1)$-cube in the subgraph G_{a+1}, and establish edges from the $(a+1)$-cube to the two (descendent) a-cubes.

Step 3: In the general case, the same procedure continues and the subcubes in subgraph G_i, $a <= i < m$, are combined to form subcubes of the subgraph G_{i+1}, until subgraph G_m is finally formed.

Example 4: Consider D2Tree of Fig. 3 (a). Fig. 3 (b) shows the final solution space G when $S_{F,n}$ is disjoined into an $S_{F,min}$.

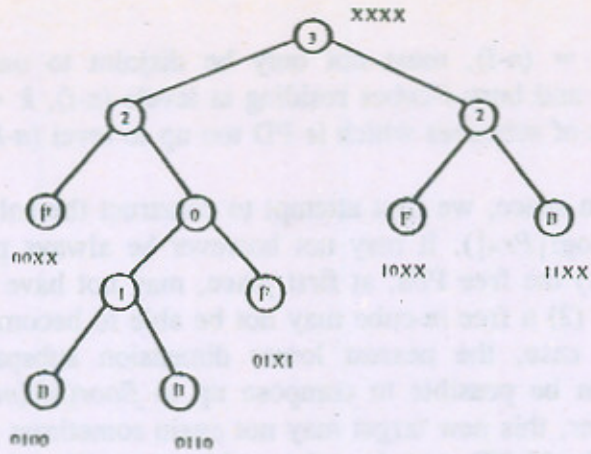


Figure 3 (a): Original D2Tree for Example 4

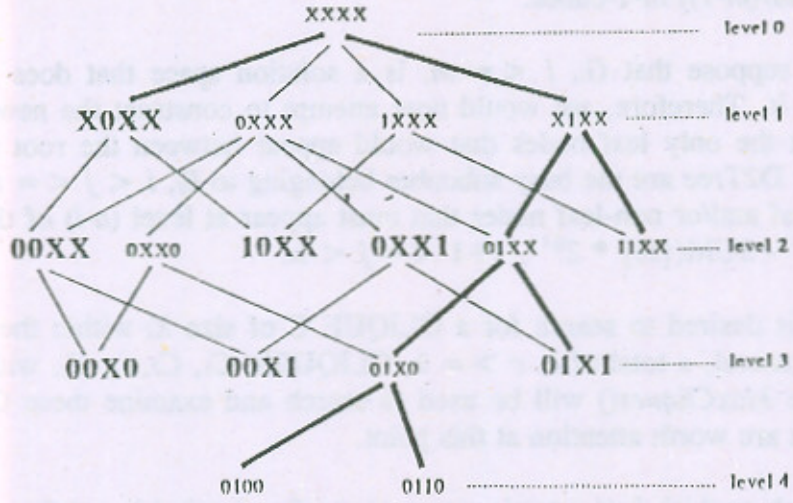


Figure 3 (b): Final solution space G when $S_{F,n}$ is disjoined into an $S_{F,min.}$, for Example 4

COMPLEXITY OF THE OPTIMAL AND NEAR-OPTIMAL SOLUTIONS

Consider the solution space $G < V, E >$ created from $S_{F,min}$ and $S_{B,n}$ as described above. Suppose $M_{F,n} = \mathbf{U}\ \{F_i\}$, $g <= i <= m$, is the optimal solution for compaction which exists in the solution space; F_m and F_g representing the largest and the smallest dimension free subcube(s) in $M_{F,n}$. Suppose $NM_{F,n} = \mathbf{U}\ (NF_i)$, $g <= i <= m$, represents the near-maximal set corresponding to the near-optimal solution. In the near-optimal solution, it would still be possible to find all $M_{F,n}$ subcube(s) of the largest dimension. That is, NF_m is equivalent to F_m. However, it may not always be possible to obtain all the lower-dimension largest possible subcubes as promised by the optimal solution. That is, in a sense, $\mathbf{U}\ (NF_i) <= \mathbf{U}\ (F_i)$, where $g <= i <= m$-1.

IMPLEMENTATION OF NEAR-OPTIMAL SUBCUBE COMPACTION

The compaction scheme is based on a recursive algorithm. However, the following important concept will be implicitly incorporated within the final solution. At all levels l, $l = 1$ down to the last level, of the solution D2Tree, all the leaf free and busy k-cubes and all the

intermediate k-cubes, $k = (n\text{-}1)$, must not only be disjoint to one an other, but, they, in conjunction with all free and busy i-cubes residing at levels $(n\text{-}i)$, $k < i <= (n\text{-}1)$, of the new D2Tree, must form a set of subcubes which is PD too up to level $(n\text{-}k)$.

Given the solution space, we first attempt to construct the solution graph up to level $(n\text{-}m)$, where $m = floor\ (\log_2|P_{F,n}|)$. It may not however be always possible to create a largest free m-cube. Because, (1) the free PEs, at first place, may not have been able to combine into an m-cube. Or, even so, (2) a free m-cube may not be able to become part of a PD solution up to level $(n\text{-}m)$; in that case, the nearest lower dimension subspace $G_{m\text{-}1}$ will have to be examined. It might then be possible to compose up to $floor(M/(m\text{-}1))$ many free m-1-cubes from the M PEs. However, this new target may not again sometimes be fully (or even partially) achievable, because (1) the M PEs may have been able to combine into only an x number of m-1-cubes, where $1 <= x <= floor(M/(m\text{-}1))$, or into no m-1-cube at all (provided there were no m-1-cube(s) in the original $S_{F,n}$), and/or (2) a PD solution up to level $n\text{-}(m\text{-}1)$ may permit only less than $floor(M/(m\text{-}1))$ m-1-cubes.

Accordingly, suppose that G_i, $i <= m$, is a solution space that does include one or more free i-cubes in it. Therefore, we would now attempt to construct the new D2Tree up to level $(n\text{-}i)$. Note that the only leaf nodes that would appear between the root node level and level $(n\text{-}i)$ of the new D2Tree are the busy subcubes belonging to B_j, $i < j <= n\text{-}1$. Therefore, the number, K_i, of leaf and/or non-leaf nodes that must appear at level $(n\text{-}i)$ of the new D2Tree is given by: $K_i = 2^{(n-i)} - \text{SUM}(|B_j| * 2^{(j-i)},\ i+1 <= j < n$.

Therefore, it is desired to search for a CLIQUE C of size K_i within the solution space G_i. Let there be, in general, a total of c, $c >= 1$, CLIQUES, C_1, C_2,..., C_c, within G_i, each of size K_i,. A procedure *MaxCliques*() will be used to search and examine these CLIQUES. The following two notions are worth attention at this point.

1. Although a G_i (such as this) does contain one or more free i-cube(s), yet there may not exist even a single CLIQUE that would incorporate such free i-cube(s). Consequently, a further lower solution space will have to be examined.

2. If there do, however, exist one or more CLIQUES, C_i, with one or more free i-cubes incorporated in them, then it is still possible that none of the CLIQUES, in conjunction with the busy subcubes belonging to $\mathrm{U}\{B_j\}$, $i+1 <= j <= n\text{-}1$, will provide a PD solution up to level $n\text{-}i$; in that case too a further lower solution space will have to be examined.

In the following, however, we assume (for the sake of explanation) there does exist at least one CLIQUE that would give a PD solution up to level $n\text{-}i$. Each CLIQUE C_i, in general, will be a union of four distinct parts: $C_{B,i}$, $C_{F,i}$, $C_{H,B,i}$, and $C_{H,i}$. Here $C_{B,i}$ represents the busy i-cubes belonging to B_i, $C_{F,i}$ represents the free i-cubes of G_i, $C_{H,B,i}$ represents the composite busy i-cubes each having more than one descendent busy subcubes, and $C_{H,i}$ represents the i-cubes having free as well as busy subcubes in their descendents. One issue need be resolved first, *i.e.*, which i-cubes of G_i must be fed as input to the procedure *MaxCliques*(). In this regard, note that each subcube belonging to B_i is already disjoint to every other subcube in G_i. Therefore, in order to minimize computational work, only the subcubes belonging to set $A = G_i$

- B_i need be passed on to procedure *MaxCliques*(). Accordingly, each output CLIQUE C_i is now supposed to consist of three parts only, *i.e.*, $C = C_{F,i} \cup C_{H,B,i} \cup C_{H,i}$, and will have size of K_i - $|B_i|$. However, heuristically, we choose, with the help of procedure *MaxCliques*(), that particular MAX-CLIQUE $C_{max} = C_{F,max} \cup C_{H,B,max} \cup C_{H,max}$ which

(i) ensures that the set of subcubes $C_{max} \cup W$, where $W = \cup \{B_j\}$, $i <= j <= n-1$, is PD up to level
(n-i); and besides,

(ii) incorporates maximum number of totally free i-cubes, $C_{F,max}$, in it.

(iii) If more than one CLIQUE qualify conditions (i) and (ii), then the one having the maximum number of composite busy i-cubes is chosen. The idea here is that, if all busy subcubes are allowed to cluster together and form the largest possible, and as many as possible, composite busy subcubes of the kind $C_{H,B,i}$, then the free subcubes, as a result, will also automatically draw closer to one another and form the largest possible, and as many as possible, subcubes of the sought after solution $NM_{F,n}$.

```
Algorithm SubcubeCompaction()
BEGIN
  create the solution graph G;
  Q = n-cube corresponding to MPn;
  SubtreeUnderQ (Q, n, M);
END

Procedure SubtreeUnderQ (Q, q, freePEs (Q))
BEGIN
    /* Let Bj', 0 <= j < q, represent the set of busy j-cubes
       that are descendents of the q-cube Q */;

    i = floor(log2 (freePEs (Q)));
   FreeCubesCreated = false;

   WHILE (NOT FreeCubesCreated AND (i > 0)) DO
      /* Let gi in Gi corresponds to those i-cubes that are
         descendents of the passed-on q-cube Q */;

  IF (gi contains at least one free i-cube) THEN

        W = U { Bj' }, i <= j < q-1;
        Ki = 2^(q-i) - SUM(|Bj'| * 2^(j-i)), i <= j < q-1;
        A = {all i-cubes of gi} - Bi';
        Cmax = MaxCliques (A, Ki, W);
```

```
        IF (Cmax is not an empty set) THEN
            FreeCubesCreated = True;
            NFi = (NFi U CF,max);

            FOR (each i-cube Q in CH,max) DO
                SubtreeUnderQ (Q, i, freePEs (Q));
        ENDIF ELSE
            i = i - 1;
    ENDIF
  END WHILE
RETURN
```

(iv) Again if there happen to be more than one CLIQUE obeying conditions (i), (ii) and (iii), then look down at subspace G_{i-1}, under/through the non-free subcubes of $C_{H,i}$ part, and retain a CLIQUE that has maximum number of free (i-1)-cubes in G_{i-1}, hanging under its $C_{H,i}$ part. This tactic would later help form largest possible and as many as possible frees subcubes of lower dimensions.

The algorithm shows that procedure *MaxCliques*() would return a desired CLIQUE only if it successfully incorporates one or more free i-cubes. A very noteworthy aspect of the solution is that a desired C_{max} is picked on the fly when all different CLIQUES of size K_i - | B_i | are 'marched past', one by one, in front of the procedure *MaxCliques*(). As soon as NF_i receives the free i-cubes belonging to $C_{F,max}$, this phase of the algorithm would end. All subcubes of G_i - B_i not contained in C_{max} are discarded. The composite busy i-cubes of $C_{H,B,max}$ part are too not processed any further, because they do not have any free PEs hanging under them.

Now let us pay attention to the remaining solution space, which comprises descendents of the i-cube(s) belonging to $C_{H,max}$. The number of remaining free PEs, now available to compose the free subcube(s) of lower dimensions, would be equal to M - (| $C_{F,max}$ | * 2^i).

COMPOSING COMPLETE $NM_{F,N}$ IN A RECURSIVE WAY

Since all subcubes belonging to C_{max} are mutually disjoint, hence each and every descendent subcube hanging under a given i-cube belonging to $C_{H,max}$ must be disjoint to all descendent subcubes of all other i-cubes belonging to $C_{H,max}$. Therefore, the above algorithm can be similarly recursively applied on (the descendents of) each i-cube of $C_{H,max}$, and so forth. This is done with the help of procedure *SubtreeUnderQ*(); wherein the *free-PEs*(Q) represents the number of free PEs rooted under the particular i-cube Q belonging to $C_{H,max}$ in question. During this processing all free j-cubes, $j < i$, found are gathered in their respective sets NF_j. This way, all the NF_j, $g <= j <= m$, when put together would form the desired solution $NM_{F,n}$. When this is over, the multiprocessor will be partitioned according to subcubes in the set $NM_{F,n}$ U $S_{B,n}$, and a new solution D2Tree will emerge.

PERFORMANCE RESULTS

Our compaction and migration scheme was implemented in C++ and run on SUN SPARC station 1+. The assumptions used in the simulation are adopted from [cs87] and [ct92]. A new request arrives every time unit. The dimensions of subcubes required by the requests are assumed to follow uniform, normal, and biased normal distributions. The subcube residency time is determined by uniform distribution [a, b]. The generation, allocation, deallocation, and queuing of requests is continued until a pre-determined time limit T is reached. A FCFS scheduling discipline is used for the requests. For each of the request size and residency time distribution, the following statistics were recorded and averaged over 30 independent runs.

C_h: Number of requests that were allocated in time T.

d_h: Average waiting delay per request.

E: Efficiency of the processor utilization.

Table-I, for example, gives performance results for medium-scale systems. For each distinct case, a comparison is given for the case (a) when no compactions are allowed at all and (b) when compactions are allowed, as and when necessary.

Table-I Performance Results for Large-scale System

N	T	Resid dist	Compctns Migratns	Avg. PEs Migraatd	Jobs assignd	Delay (d_h)	Utilztn (%)
256	200	[4,10][1]	None	None	142.53	19.4	78.28
			2.96/2.25	21.48	148.73	18.31	81.06
		[6,16][2]	None	None	155.53	15.83	73.31
			5.52/6.33	17.38	174.03	10.5	82.45
		[12.5,20,5][3]	None	None	155.26	15.46	70.11
			6.22/4.62	15.05	172.93	11.47	78.20
1024	200	[5.9,11.7][4]	None	None	142.3	18.59	76.75
			3.12/1.77	76.68	148.9	18.24	79.04
		[6,29][5]	None	None	134.53	18.99	69.87
			5.42/5.75	60.21	158.7	14.72	82.53
		[8,32][6]	None	None	164.63	12.02	62.38
			9.3/3.45	61.49	185.5	7.32	70.82

REFERENCES

[cs87] M. S. Chen and K. G. Shin, Processor Allocation in n-cube Multiprocessor using gray codes. IEEE Trans. On Computers, C-36: 1396-1407, Dec. 1987.

[js87] M. Jeng. Tolerance and Dynamic Partitioning in Large-Scale Parallel Systems. PhD thesis, Purdue University, 1987.

[hj89] H. J. Siegal et al. Using the Multistage Cube Topology in Parallel Supercomputers. IEEE Proceeding, Dec. 1989.

[js90] M. Jeng and H.J. Siegal, A distributed management scheme for partitionable parallel computers. IEEE Transactions on Parallel and Distributed Systems, Jan. 1990.

[lw86] W. Lin and C.-L. Wu. Reconfiguration Procedures for a Polymorhpic and Partitionable Multiprocessor, IEEE Trans. On Computers, C-35: 910-916, Dec. 1982.

[mna2] M. N. Ayyaz. Subcube Migration in Multistage Interconnection Networks, Part-II, this issue.

Table 2: Performance Results for Medium-scale Systems

N	T	Resid dist	Compctns performd	Jobs assignd	delay (d_h)	Utilizatn (%)
64	200	$[3,8]^1$	None	144	19.29	80.57
			2.8	146.70	18.98	81.56
		$[4, 11]^2$	None	156.73	16.46	77.16
			8.63	162.47	15.46	79.43
		$[8,14.5]^3$	No	149.33	17.34	73.76
			12.01	155.86	16.11	76.51
128	200	$[3.3,8.7]^1$	None	147.60	18.11	79.24
			3.65	150.60	17.96	80.47
		$[4.5, 14.2]^2$	None	153.13	17.29	75.45
			10.90	159.50	15.75	78.59
		$[9,16.5]^3$	No	161.66	14.85	70.77
			15.68	167.70	13.42	73.40