Abstract. The n-dimensional hypercube or n-cube is a highly concurrent multiprocessor architecture consisting of 2^n nodes each connected to n neighbors. Such machines have been advocated as ideal ensemble architectures because of their powerful interconnection. In this paper we examine the hypercube architecture from the graph theory point of view and we consider those features which make the hypercube interconnection so appealing. Among other things, we propose a theoretical characterization of the hypercube as a graph and show how to map various other architectures into a hypercube.

Topological Properties of Hypercubes

Youcef Saad and Martin H. Schultz

Research Report YALEU/DCS/RR- 389
June 1985

This work was supported in part by ONR grant N00014-82-K-0184 and in part by a joint study with IBM/Kingston

1. Introduction

The hypercube is a loosely coupled parallel multiprocessor based on a binary n-cube network and introduced under different names (for example, cosmic cube, n-cube, binary n-cube, and boolean n-cube). A few machines based on the hypercube topology have been implemented by several groups, see [7] for references, and others are now being built. An n-cube parallel multiprocessor consists of 2^n identical processors, each provided with its own sizable memory, and connected to n neighbors in the form of of a binary n–cube network.

There are essentially two broad classes of MIMD parallel multiprocessor processor designs with large number of processors. The first type of architecture consists of a large number of identical processors conneted to one another according to some convenient pattern. In these machines there is no shared memory and no global synchronization. Moreover, communication is achieved by message passing and computation is data driven (some designs incoporate a global bus which does not constitute the main way of intercomunication). By message passing we mean that data or possibly code are transferred from processor A to processor B by traveling accross a sequence of nearest neighbor nodes starting with node A and ending with B. Synchronization is driven by data in the sense that computation in some node is performed only when its necessary data is available. Examples include grid networks such as the finite element machine [1], tree machines [3], and the cosmic cube [7]. At the border line of this class we might also include the dataflow machines which utilize the same concept of data driven synchronization but adopt a more dynamic way of circulating data. The main advantage of such architectures, is their simplicity of design. The nodes are identical, or are of a few different kinds, and can therefore be fabricated at relatively low cost.

The second important class of parallel multiprocessors consists of those systems with N identical processors connected via a large switching network to N memories. Thus the memory can be viewed as split into N "banks", and shared between the N processors. Variations on this scheme are numerous, but the essential feature here is the switching network. Examples include the Ultra computer developed at NYU [5] which uses an Omega network. The main advantage of this second configuration is that it makes data access transparent to the user who may regard data as being held in a large memory which is readily accessible to any processor. This greatly facilitates the programing of the machine but memory conflicts can lead to degraded preformance. Also, the network can simulate any of the communication patterns of the first type of architectures. However, switched network models cannot easily take advantage of proximity of data in problems where communication is local. Moreover, the switching network becomes exceedingly complex to build as the number of nodes increases. In fact, to connect N nodes the Ultracomputer requires a total of $O(Nlog_2N)$ identical 2 x 2 switches. This raises the problem of reliability as the probability of failure increases proportionally to the number of components. To be sure, switched models can easily be made fault tolerant by shutting down failing nodes. Unlike shared memory models, the decision of shutting down failing nodes and choosing alternate routes is a local one.

As previously mentioned, one of the most important advantage of fixed connection designs is the ability to exploit particular topologies of problems or algorithms in order to minimize communication costs. Thus a two-dimensional grid network is ideally suited for solving discretization of elliptic partial differential equations, e.g., by assigning each grid point to its counterpart in the array, because iterative methods for solving the resulting linear systems, require only nearest neighbor grid-point interaction. This means that if a general purpose ensemble architecture is to be useful for solving partial differential equations it must have powerful mapping capabilities, i.e., it must be possible to easily map many common geometries such as grids or linear arrays into the architecture. The hypercube is a machine of the first class which has excellent mapping capabilities.

This explains in part the current growing interest in the parallel multiprocessor machines based on the Boolean n-cube.

It is the purpose if this paper to study the topological properties of the hypercube. We will first derive some simple properties of the hypercube regarded as a graph and will propose a theorem that will describe a hypercube by a few characteristic properties. Mapping other topologies into hypercubes is very important for designing efficient algorithms that map perfectly into hypercubes. An interesting feature of the hypercube is the ease with which data exchange between processors can be organized thanks to some useful properties of the binary numbering of its nodes. We will consider this problem in detail and show how to map rings, linear arrays, multi-dimensional meshes, and trees into hypercubes.

2. The hypercube graph and its basic properties

In what follows, we regard the hypercube or n–cube as a graph and often use the terms vertices or nodes interchangeably for the processors they represent. A 3–cube can be represented as an ordinary cube in three dimensions whose vertices are the $8 = 2^3$ nodes of the 3–cube, see Figure 1. More generally we can construct an n-cube as follows. First the 2^n nodes are labeled by the 2^n binary numbers from 0 to $2^n - 1$. Then a link between two nodes is drawn if and only if their binary numbers differ by one and only one bit. In this paper we will refer to the hypercube or n–cube graph, or hypercube or n–cube, as the graph thus defined.

Definition 2.1. An n-cube graph is an undirected graph consisting of $k = 2^n$ vertices labeled from 0 to $2^n - 1$ and such that there is an edge between any two vertices if and only if the binary representations of their labels differ by one and only one bit.

The first important property of the n-cube is that it can be constructed recursively from lower dimensional cubes. More precisely consider two identical $(n-1)$-cubes whose vertices are numbered likewise from 0 to 2^{n-1}. By joining every vertex of the first $(n-1)$-cube to the vertex of the second having the same number, one obtains an n-cube. Indeed, it suffices to renumber the nodes of the first cube as $0 \wedge a_i$ and those of the second by $1 \wedge a_i$ where a_i is a binary number representing the two similar nodes of the $(n-1)$-cubes and where $\wedge$ denotes the concatenation of binary numbers. This is illustrated for $n = 4$ in Figure 2, where a 4-cube is obtained by joining all corners of an inner 3-cube with the corresponding corners of an outer 3-cube. An interesting geometric property of the illustration is that it provides one way of constructing higher dimensional cubes, from 3-cubes by simply repeating the above process with more enclosing cubes.

Separating an n-cube into the subgraph of all the nodes whose leading bit is 0 and the subgraph of all the nodes whose leading bit is 1, the two subgraphs are such that each node of the first is connected to one node of the second. If we remove the edges between these two graphs we get 2 disjoint $(n-1)$-cubes. This operation of splitting the n-cube into two $(n-1)$-cubes so that the nodes of the two $(n-1)$-cubes are in a one–to–one correspondance will be refered to as *tearing*. The tearing suggested above gives privilege to the leading bit but there is no particular reason for this. More generally, for a given numbering, tearing simply amounts to separating the graph into two subgraphs obtained by considering all the nodes whose i^{th} bit is 0 and those whose i^{th} bit is 1. This will be refered to as tearing along the i^{th} direction. Since there are n bits, there are also n directions. These simple properties are summarized in the following proposition.

Proposition 2.1. *There are n different ways of tearing an n-cube, i.e., of splitting it into two $(n-1)$–subcubes so that their respective vertices are connected in a one-to-one way. Given the labeling of Definition 2.1, each different tearing corresponds to splitting the n-cube graph into two*

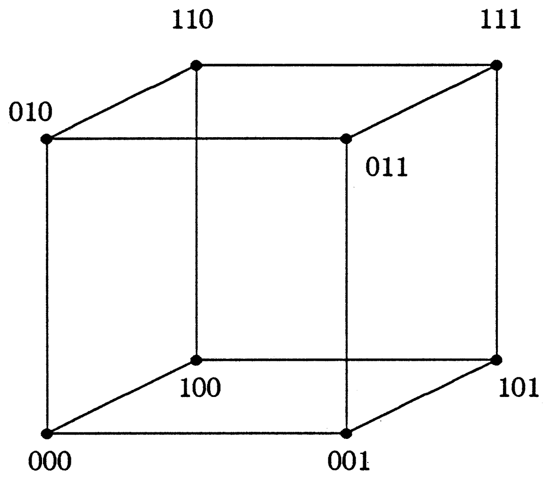


Figure 1: 3-D view of the 3-cube.

subgraphs: one whose node labels have a one in position i and one whose node labels have a zero on position i.

The following result tells us how many ways there are for labelling an n–cube.

Proposition 2.2. *There are $n!2^n$ different ways in which the 2^n nodes of an n-cube can be numbered so as to conform with Definition 2.1.*

Proof. The proof is by induction. The result is trivial for $n = 0$. Assume it is true for $n - 1$. To number the nodes of the n-cube we will first choose their leading bits to be either zero or one. To do so, we wil tear the n-cube into two $n - 1$ cubes (there are n different ways to do it). Then we number the nodes of the first $(n - 1)$-cube (in $(n - 1)!2^{n-1}$ different ways) and add a one as leading bit, and the nodes of the second cube in the same way and then add a zero as leading bit. A second numbering is obtained by reversing the bits zero and one. We thus obtain a total of

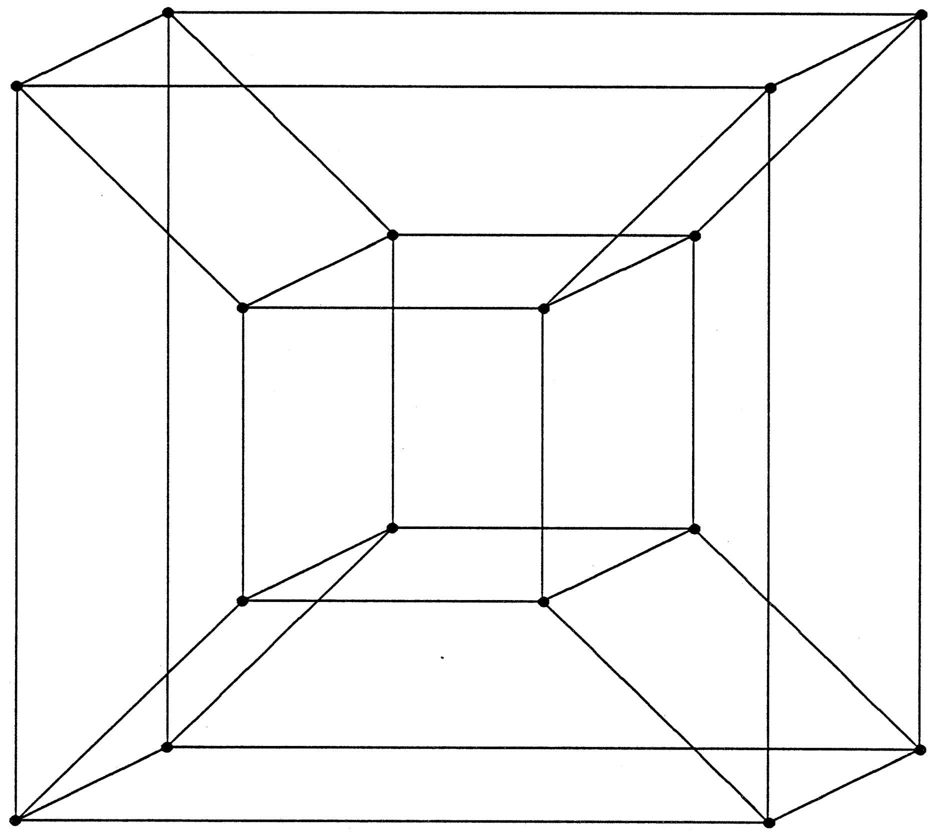
$$n[(n-1)!2^{n-1} + (n-1)!2^{n-1}] = n!2^n$$

different numberings of the vertices of the n-cube.

∎

Note that without the restriction that the numbering must be conform to the Definition 2.1 we would have a total of $(2^n)!$ different ways of numbering 2^n different vertices, a much larger number than that of Propostion 2.2.

Proposition 2.1 has the following important consequence.



due ipercubi 3d formano un ipercubo 4d collegando gli estremi corrisopndenti

Figure 2: 3-D view of the 4-cube.

Proposition 2.3. *Any two adjacent nodes A and B of an n-cube are such that the nodes adjacent to A and those adjacent to B are connected in a one-to-one fashion.*

Proof. Since A and B are neighbors, their node numbers differ by one bit, say the i^{th} bit. Let us tear the n–cube along the the i^{th} direction. Then the binary numbers of the neighbors of A and those of B differ only in their i^{th} bit and hence they are connected in a one–to–one fashion.

∎

We can define the parity of a node to be positive if the number of 1's in its binary label is even and negative otherwise. It is clear that neighboring nodes have opposite parity. The following result follows from this property.

Proposition 2.4. *There are no cycles of odd length in an n-cube.*

Proof. Consider a cycle including node A. $A = A_1, A_2, \ldots, A_t = A$. As we travel from node A_i to node A_{i+1}, $1 \le i \le t-1$, the parity changes. Since $A_1 = A_t$, there must be an even number of changes or t must be odd or the the length of cycle must be even.

∎

Given any two nodes of an n-cube there is always a path between them. Given a labelling of the n–cube one way of reaching node B from node A is to modify the bits of the label of A one at a time in order to transform the binary label of A into the binary label of B. Each time one bit is changed this means that we have crossed one edge. This provides a simple way of constructing a path of length ***at most*** n between any two vertices of an n-cube. Therefore, recalling that the diameter of a graph is the maximum distance between any two nodes in the graph, we can state that

Proposition 2.5. *The n-cube is a connected graph of diameter n.*

The above propositions establish some basic properties of the n-cube as a graph. The important question we would like to answer next is how to recognize a n–cube in a simple way, i.e., how to characterize an n-cube by a few simple rules. As an example of application, looking at a four by four grid with nearest neighbor connection and wrap-around at the edges (of the grid) one might ask whether the corresponding graph is an n-cube, i.e., whether its 16 nodes can be numbered according to the rule of the definition 2.1. It is clear that without the wrap around at the edges the grid cannot be a cube because in a cube all vertices have the same degree. The next result answers this question.

Theorem 2.1. *A graph $G = (V, E)$ is an n-cube if and only if*

1. *V has 2^n vertices;*
2. *Every vertex has degree n;*
3. *G is connected;*
4. *Any two adjacent nodes A and B are such that the nodes adjacent to A and those adjacent to B are linked in a one–to–one fashion.*

Proof. Necessary condition: Conditions 1 to 4 are clearly satisfied for an n-cube as a result of the definition and some of the previous propositions.

Sufficient condition: The proof is by induction. It is clear that the property is true for $n = 1$. Assume that it is true for $n-1$, i.e., that any graph having 2^{n-1} nodes satisfying properties 1 to 4, is an $(n-1)$-cube. The proof consists in separating the graph in two subgraphs each of which has the same properties for $n-1$.

Consider any two adjacent nodes R (for red) and B (for black) of the graph. According to Property 4, the neighbors of R and those of B are connected in a one to one fashion. We can therefore color the neighbors of the red node (except the one node which is already black) in red and the neighbors of the black node (except the one node which is already red) in black. This process can be continued until exhaustion of all links. We refer to two nodes of different colors that are linked by an edge as two opposite nodes.

After this is done we have :

(a) All the nodes have been colored either black or red. This is because the graph is connected and therefore there is a path between the original node R (or B) to any node.

(b) Exactly half the nodes have the color red and the other half the color black, because all the black nodes and the red nodes are linked in a one–to–one fashion.

(c) It is clear that the red nodes constitute a connected graph, since by contruction each node is connected to the original R node. Same property holds for the B nodes.

(d) Consider the two subgraphs obtained by removing all red–black links. Thus each node loses exactly one edge, i.e., its degree is $(n-1)$. (In graph theory terminology the set of red–black edges is called a cut set.) Then Property 4 is satisfied for the subgraph of the red nodes (resp. the black nodes).

(e) Because of Property 4 and the preceeding construction two red nodes are adjacent if and only their black opposites are adjacent.

By the induction hypothesis and by (b), (c), (d) and (e) the subgraph of the red nodes is an $(n-1)$-cube. Now label the red nodes according to the definition and use the same labeling for the black nodes opposite to them (we can use the same labeling thanks to (e)). Adding the bit zero in front of the red nodes and the bit one in front of the black nodes we obtain a labeling of the nodes of the initial graph.

∎

A consequence of Theorem 2.1 is that the 4 x 4 grid with wrap-around at the edges (often referred to as the torus) is a 4-cube. A generalization is the 4 x 4 x 4 grid in 3 dimensions with again wrap-around at the edges. From Theorem 2.1 this is nothing but a 6-cube. Thus, these mappings provide simple three dimensional geometric representations of n-cubes when $n \leq 6$.

3. Distances and paths in the Hypercube

Any multiprocessor system must allow for data to be exchanged among all of its nodes. Let A and B be any two nodes of an n-cube and consider the problem of sending data from node A to node B. The way in which this is achieved is to move the data (ideally in packets) along a path from A to B crossing a (possibly small) number of processors. By definition, the length of a path between two nodes is simply the number of edges of the path. As was already mentioned in section 2, there exists a path of length at most n between any two nodes. To reach B from A it suffices to cross successively the nodes whose labels are those obtained by modifying the bits of A one by one in order to transform A into B. Assuming that the binary labels for A and B differ only in i bits i.e, the Hamming distance between the labels for A and B, $H(A,B) = i$, the length of the path will be i. Clearly, there is no path of smaller length between the nodes A and B. This elementary result can be formalized as follows.

Proposition 3.1. *The minimum distance between the nodes A and B is equal to the number of bits that differ between the labels for A and B, i.e., to the Hamming distance $H(A,B)$.*

For future reference we would like to explicitly write down one of the paths suggested by the proof of the above proposition. This path corresponds to correcting the first different bit in A and B, then the second and so on to the last bit different in A and B. Let $A \equiv a_1a_2..a_n$ and $B \equiv b_1b_2..b_n$ be the labels of A and B where a_i and b_i are the bits zero or one. For convenience we assume without loss of generality that A and B differ in their i leading bits. Then one path from A to B is the following

$$\begin{aligned}
A = \text{node } 0 &= a_1a_2a_3\ldots a_ia_{i+1}\ldots a_n;\\
\text{node } 1 &= b_1a_2a_3\ldots a_ia_{i+1}\ldots a_n;\\
\text{node } 2 &= b_1b_2a_3\ldots a_ia_{i+1}\ldots a_n; .\\
\ldots\ \ &\ldots\\
B = \text{node i} &= b_1b_2..b_ia_{i+1}\ldots a_n
\end{aligned}$$

The generalization to the case where the bits different in A and B are not necessarily the leading ones is straightforward.

One important question we would like to address is whether or not there are different paths between A and B. The existence of such paths might be useful for speeding-up transfers of large amounts of data between two nodes. It also provides a way of selecting alternative routes in case a given node in a path has failed. [2]. In order for this to be possible the paths must not cross each other, i.e., they must not have common nodes, except for nodes the A and B. We will refer to such paths as *parallel paths*. So the above question can be reformulated as follows: *How many parallel paths are there between any two nodes A and B?*

A simple look at the above path between A and B reveals that there is no reason why to start by correcting the first bit different in A and B. More generally, assuming again that the i bits different in A and B are in front, one might start correcting the j^{th} bit, where $1 \leq j \leq i$, then the $(j+1)^{st}$ bit and so forth until the i^{th} bit is reached, after which we correct in turn bits $1, 2, \ldots (j-1)$. We can thus define i different paths and number them from $j = 1$ to $j = i$. It is easy to prove that any two such paths are parallel. Indeed, the label h, of the node X_h of any path $X_0, X_1, \ldots X_h, \ldots, X_i$, (with $X_0 \equiv A$) of the above i paths is different from the label of A in exactly h bits. By construction, any two different paths starting the correction in positions j_0 and j_1 respectively, cannot reach the same node X_h in the same number of steps. Moreover, they cannot reach this same node in two different numbers of steps.In fact if they did, one path would correct A into X_h by changing l_1 bits while the other will achieve the same result by changing l_2 bits with $l_1 \neq l_2$, which is a contradiction. Therefore, we can state the following proposition.

Proposition 3.2. *If A and B are any two nodes in an n-cube then there are $H(A,B)$ parallel paths of length $H(A,B)$ between the nodes A and B.*

Note that there are many different ways of choosing the $H(A,B)$ parallel paths. In other words there is no uniqueness of the set of i parallel paths. Also observe that when $i = n$, the result is optimal in that we can use the maximum allowable number of paths leaving from node A, since the degree of any node is n. We would like now to improve the above result by showing that if we relax the restriction that the length must be i then as many as n parallel paths can be found even for the case $i<n$. This is important as it will allow us to use the *full bandwidth of the multiprocessor* for transfering data between two given processors.

Proposition 3.3. *Let A, B be any two nodes of an n-cube and assume that $H(A,B)<n$. Then there are n parallel paths between A and B. Moreover, the length of each path is at most $H(A,B)+2$.*

Proof. In addition to the i paths described prior to proposition 3.2 and numbered from 1 to i, consider the paths which we will number from $j = i+1$ to n, obtained as follows. First, modify the bit a_j into its complement $\bar{a}_j$. Thus the additional paths start with

$$\text{node} \quad 1\text{:} \quad \hat{a}^{(j)} = a_1 a_2 \ldots a_i a_{i+1} \ldots \bar{a}_j a_{j+1} \ldots a_n.$$

as their first node after the starting node. We then correct bits 1 through i according to *one of the i paths* of the Proposition 3.2 to reach, after i steps, the node

$$\text{node i+1:} \quad \hat{b}^{(j)} = b_1 b_2 \ldots b_i b_{i+1} \ldots \bar{a}_j a_{j+1} \ldots a_n.$$

Finally we change the bit $\bar{a}_j$ back to a_j to reach the final destination B. It is clear by construction that the additional paths thus defined will never cross each other and that they will not cross any of the initial i paths. Morever the length of each of the additional paths is $i+2$.

∎

Note that we can even say that there are $H(A,B)$ paths of length $H(A,B)$ each and $n-H(A,B)$ paths of length $H(A,B)+2$ each. What the proof indicates is that the first $H(A,B)$ paths do not use all of the tearings of the cube. The additional paths exploit the unused $(n - H(A, B))$–cubes corresponding to the bits in labels of nodes A and B which agree.

4. Mapping other geometries into the Hypercube

In this section we will be concerned with the problem of mapping other topologies (rings, meshes, and trees) into hypercubes. By mapping other geometries into a hypercube we mean the following. We are given some graph $G = (V, E)$ having no more than 2^n vertices and we would like to assign the vertices of the graph into the nodes of the n-cube so that every adjacent vertex of the graph belongs to neighboring nodes of the n-cube. There are mainly two different reasons why such mappings are important:

1. Some algorithm may be developed for another architecture for which it fits perfectly. Then one might wish to implement the same algorithm with little additional programming effort. If the original architecture can be mapped into the hypercube this will be easy to achieve.
2. A given problem may have a well defined structure which leads to a particular pattern of communication. Mapping the structure may result in substantial savings in communication time. A good example is that of mesh geometries that arise from the dicretization of elliptic partial differential equations in one, two or three dimensions. Most iterative methods for solving elliptic PDE's require only local communication, i.e., communication between mesh points that are neighbors. If the mesh is perfectly mapped into a hypercube then only local communication will be required between the nodes of the hypercube thus resulting in important savings in transfer times.

In this section, we consider mapping rings grids and tree structures into hypercubes.

4.1. Mapping rings and linear arrays into hypercubes

Given a ring–structured graph of 2^n vertices, we consider the problem of assigning its vertices to the nodes of an n–cube in such a way as to preserve the proximity property, i.e., so that any two adjacent vertices belong to neighbor nodes. Another way of viewing this problem is that we seek for a cycle of length $N = 2^n$ that crosses each node once and only once. In graph theory terminology, we are looking for a Hamiltonian circuit in the n–cube.

If we number the nodes of the n–cube according to Definition 2.1, i.e., so that two neighboring nodes differ by one and only one bit, a Hamiltonian circuit simply represesents a sequence of n-bit binary numbers such that any two successive numbers have only one different bit and so that all binary numbers having n bits are represented. Binary sequences with these properties are called Gray codes, and have been extensively studied in coding theory, see e.g., [6].

There are many different ways in which Gray codes can be generated, but one of the best known methods leading to the so-called binary reflected Gray code is as follows. We start with the sequence of the two one-bit numbers 0 and 1. This is a one–bit Gray code. To built a two-bit Gray code take the same sequence and insert a zero in front of each number, then take the sequence in *reverse order* and insert a one in front of each number. In other words we get the sequence

$$G_2 \equiv \{00, 01, 11, 10\}.$$

We can then repeat the process to build a 3-bit Gray code by taking the above sequence inserting a zero in front, then taking the reverse sequence and inserting a one in front:

$$G_3 \equiv \{000, 001, 011, 010, 110, 111, 101, 100\}. \tag{4.1}$$

More generally, denoting by G_i^R the sequence obtained from G_i by reversing its order, and by $0G_i$ (resp. $1G_i$) the sequence obtained from G_i by concatenating a zero (resp., a one) in front of each element of the sequence, then Gray codes of arbitrary order can be generated by the recursion:

$$G_{n+1} = \{0G_n, 1G_n^R\}. \tag{4.2}$$

It is easy to verify that such sequences are Gray codes [6].

Gray codes allow us to map rings whose lengths are powers of two into hypercubes. Suppose now that we have a ring of arbitrary length l which we would like to map into the hypercube. First observe that the mapping is possible only when l is even since according to Proposition 2.4 a hypercube does not admit odd cycles. Therefore assume that $4 \leq l \leq 2^n$. The problem is to find a cycle of length l in the n-cube, where l is even.

Let $m \equiv (l-2)/2$ and denote by $G_{n-1}(m)$ the partial $(n-1)$-bit Gray code consisting of the first m elements of G_{n-1}. Then using the above notation, a cycle having the desired property is the following

$$\{0G_{n-1}(m), 1G_{n-1}(m)^R\}.$$

Observe that when $l = 2^n$ we obtain as a particular case the formula (4.2). Therefore, we have proved the following result.

Proposition 4.1. *A ring of length l can be mapped into the n-cube when l is even and $4 \leq l \leq 2^n$.*

Finally, we point out that there is no difficulty in embedding a linear array instead of a ring, into the n-cube. It suffices to map the nodes of the linear array $P_0, P_1, \ldots P_l$ of arbitrary length $l \leq 2^n - 1$, successively into the nodes $g_0, g_1, \ldots g_l$. Given a linear array of arbitrary length l, the smallest dimension n-cube into which it can be mapped is clearly the cube of dimension $n = \lceil Log_2(l+1) \rceil$.

4.2. Mapping grids into hypercubes

One of the most attractive properties of the boolean n-cube, is that meshes of arbitrary dimensions can be imbedded in it. This is one of the main reasons for the success of the hypercube architecture. Consider an $m_1 \times m_2 \ldots \times m_d$ mesh in the d-dimensional space R^d and assume that the mesh size in each direction is a power of 2, i.e., it is such that $m_i = 2^{p_i}$. Let $n = p_1 + p_2 + \ldots p_d$ and consider the problem of mapping the mesh points into the n-cube, one mesh point per node. Observe that we have just enough nodes to accomodate one mesh point per node.

What is meant by a mapping of the mesh into the cube is an assignment of the mesh points into the nodes of the cube so that *the proximity property is preserved*, i.e., so that two neighbor points of the mesh are assigned to neighbor nodes in the cube. In the case $d = 1$ the problem was solved in the previous section by using Gray codes. We show next how to extend the ideas of the previous section to more than one dimension.

Our argument is best illustrated by an example. Consider a 2–dimensional 8 x 4 mesh i.e. $d = 2, p_1 = 3, p_2 = 2, n = p_1 + p_2 = 5$. A binary number A of any node of the 5-cube can be regarded as consisting of two parts: its first three bits and its last two bits, which we write in the form

$$A = b_1 b_2 b_3 \; c_1 c_2$$

where b_i and c_j are zero or one. It is clear from the definition of an n-cube that when the last two bits are fixed then the resulting 2^{p_1} nodes form a p_1-cube (with $p_1 = 3$). Likewise, whenever we fix the first three bits we obtain a p_2-cube. The mapping then becomes clear. Choosing a 3-bit Gray

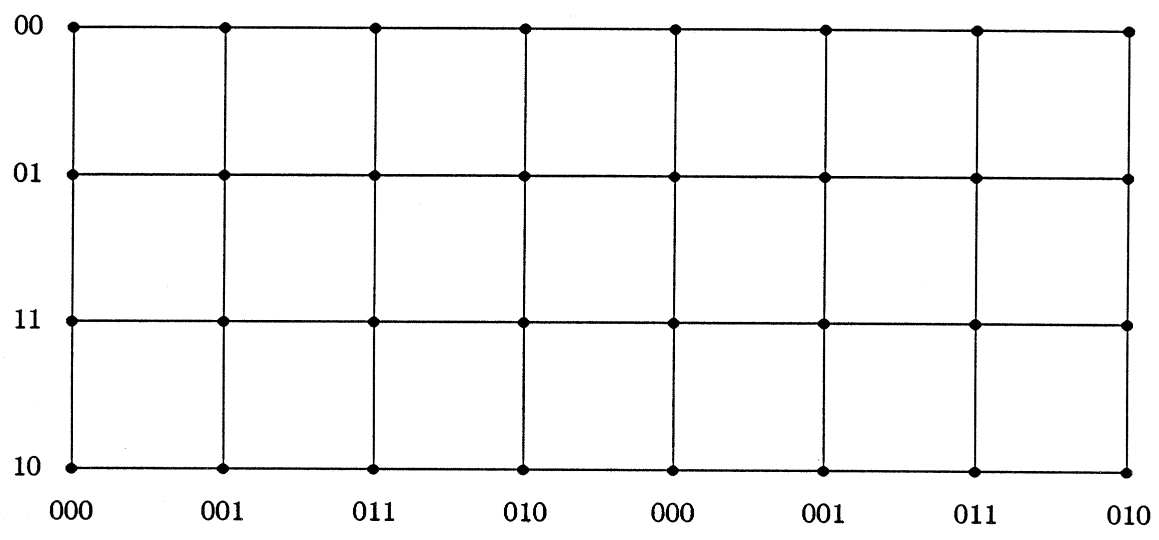


Figure 3: Two dimensional Gray-code for an 8 x 4 grid

code for the x-direction and a 2-bit Gray code for the y-direction, the point (x_i, y_j) of the mesh is assigned to the node $b_1b_2b_3\ c_1c_2$ where $b_1b_2b_3$ is the 3-bit Gray code for x_i while c_1c_2 is the 2-bit Gray code for y_j. This mapping is illustrated in Figure 3 where the binary node number of any grid-point is obtained by concatenating its binary x-coordinate and its binary y-coordinate.

Thus, if we call a Gray sequence any sub-sequence of a Gray code, we observe that any column of grid points forms a Gray sequence and any row of grid points forms a Gray sequence. Therefore we will refer to the codes defined above as 2-D Gray codes.

Generalizations to higher dimensions are straightforward and one can state the following general result.

Theorem 4.1. *Any m_1 x m_2... x m_d mesh in the d-dimensional space R^d, where $m_i = 2^{p_i}$ can be mapped into an n-cube where $n = p_1 + p_2 + \ldots p_d$ such that the numbering of the grid points has the property that its restriction to each i^{th} variable is a Gray sequence.*

Note that the assumption that all $m_i's$ are powers of 2 is not essential and the theorem can be generalized by using the remark following Proposition 4.1 concerning mapping general one-dimensional meshes. In particular, it suffices to redefine p_i in Theorem 4.1 as $p_i \equiv Log_2(m_i)$.

4.3. Mapping trees into hypercubes.

Since the hypercube is a connected graph, a well–known result of graph theory states that it contains a spanning tree, i.e., a sub–tree of the graph which contains all the vertices. For the n-cube, one way of generating a spanning tree is as follows. At the first level we put the nodes 0^n and $0^{n-1}1$, where the powers refer to concatenation. Thus the tree structure proposed can be viewed as having two connected roots. More generally, at i^{th} level we put the nodes whose bits 1

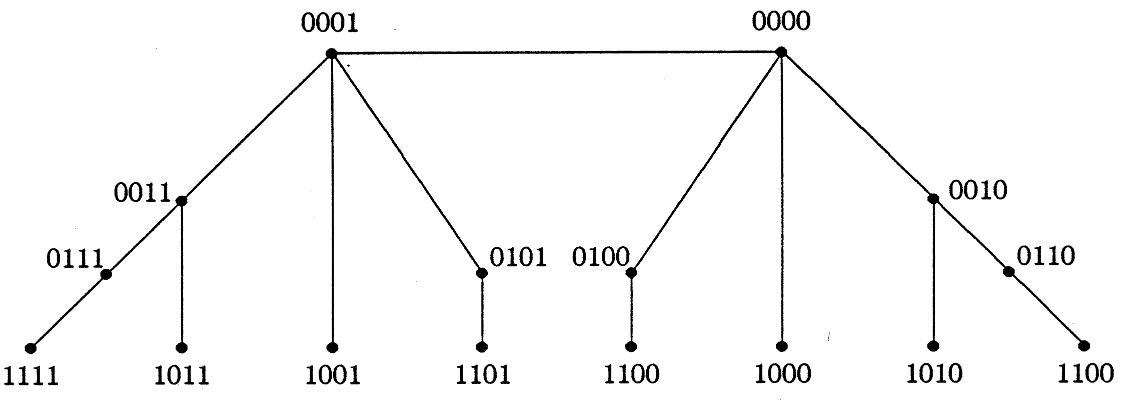


Figure 4: Tree structure of the 4-cube

through $i-1$ are zero while the bit in the i–th place is one. There are 2^{i-1} such nodes, for $i>1$. Each of these nodes is a child of one node of the previous levels, namely the node obtained by altering the i^{th} bit from one to zero. The resulting tree is depicted in Figure 4, for the case $n = 4$. The height of the tree is n. With this construction each node of level i will spawn one child of level $i+1$, one child of level $i+2$ and so on. Thus the two roots of the tree have degree n, the nodes of level 2 have degree $n-1$, the nodes of level 3 have degree $n-2$ and so on. Some other properties of the spanning tree thus defined are the following :

- A spanning tree can be generated from any two neighbors of the cube as its two roots. The structure of each of the trees is identical.
- To obtain the spanning tree of the cube of lower dimension just remove the leaves and their connections.
- The nodes of every level form a hypercube of lower dimension. To build the cube from the spanning tree all we have to do is to add the missing connections between the nodes *of the same level.*

An interesting question is whether there are other types of spanning trees that can be mapped into a hypercube. The answer is yes and there are many ways of doing so. As an example, one can first seperate the n-cube into two disconnected $(n-1)$-subcubes, generate the above spanning tree structures for each of the subcubes, and finally add in *one* of the connections between the two subcubes. We thus obtain a different spanning tree for each of the connections added.

Since we are particularly interested in relating hypercubes to other types of architectures, an important related question we wish to answer is whether it is possible to map a *binary tree* into an n-cube. Without further clarification the answer to the question may seem trivial since the number of nodes in a tree is of the form $2^n - 1$, while for the cube it is 2^n. However, the next reasonable alternative is to remove one node from the cube and try to map an n-level binary tree into the subgraph thus obtained. We show that there is no such mapping.

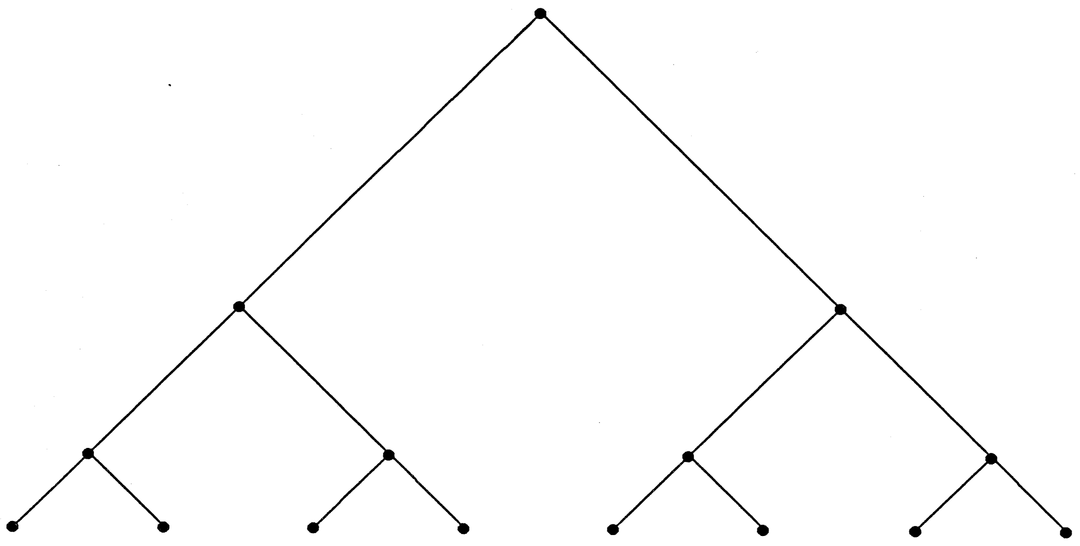


Figure 5: A 4-level binary tree

Theorem 4.2. *For $n \geq 3$ it is impossible to map a n-level binary tree into the subgraph obtained by removing one of the nodes of the n-cube graph.*

Proof. The proof consists in showing that it is not possible to construct a cube from an n-level binary tree by adding the missing node and as many adges as is necessary to make the degree of each node equal to n, as is required by the topology of the cube. Let us start with an n-level binary tree, and create one extra node. We will now assume that we can add edges between the nodes to satisfy the degree n property of the vertices of a cube. This will lead to a contradiction.

∎

We will extensively make use of the fact that there are no odd cycles in an n-cube as established by Proposition 2.4. We consider the n-level tree shown in Figure 5 and the 2^{n-1} nodes of its lowest level, i.e., its leaves. Observe that we cannot connect a leaf to another leaf because we would form an odd cycle: going up from one leaf to its parent and then going back down to the other leaf requires an even number of edges, adding the edge between the two leaves we would obtain a cycle of odd length. Therefore, all edges from the leaves must go either to higher levels or to the created node. Moreover, not all higher level nodes are possible for such edges. It is easy to show by the same argument as above that if we connect two nodes of two even or odd levels we would get an odd cycle. Therefore, the leaves can be connected only to nodes of the levels $n-1, n-3, n-5, \ldots 1+\mathrm{Mod}(n,2)$, or to the created node.

We first prove the result for the case where n is even. Let us count the number of extra edges that must leave the last level. We have 2^{n-1} nodes and each of them has only one edge. Therefore, we must add to each node a total of $n-1$ edges to make it of degree n. Hence the total number of extra edges that must leave the nodes of level n to either other levels or to the created node is:

$$n_l = (n-1)2^{n-1} \tag{4.3}$$

Let us on the other hand count how many edges can be "accomodated" by the other nodes.

- The created node can accomodate n edges.
- Each node of the levels $n-(2j-1)$ has already 3 edges, except when $j = n/2$ which corresponds to the level of the root which has only two edges. Therefore, each level can accomodate a total of $(n-3)2^{n-2j}$ when $j \neq n/2$ and $1+(n-3)2^{n-2j}$ when $j = n/2$.

Adding the number of possible accomodations we find a total of

$$\begin{aligned} n_a &= n+1+\sum_{j=1}^{n/2}(n-3)2^{n-2j} \\ &= n+1+(n-3)\sum_{i=0}^{n/2-1} 4^i \\ &= n+1+(n-3)\frac{4^{n/2}-1}{3}. \end{aligned}$$

Hence,

$$n_a = n+1+(n-3)\frac{2^n-1}{3}. \tag{4.4}$$

Let us calculate $n_a - n_l$.

$$\begin{aligned} n_a - n_l &= n+1+(n-3)\frac{2^n-1}{3} - (n-1)2^{n-1} \\ &= n+1-\frac{n-3}{3}+2^n\left[\frac{n-3}{3}-\frac{n-1}{2}\right] \\ &= \frac{n+3}{3}\left[2-2^{n-1}\right] \end{aligned}$$

For $n>2$ the number of possible accomodations is less than the number of edges that must leave from the last level, which is a contradiction.

Consider now the case where n is odd and of the form $n = 2q+1$. Evaluating evaluating n_a, we find that

$$\begin{aligned} n_a &= n+\sum_{j=1}^{q}(n-3)2^{n-2j} = n+(n-3)\sum_{j=1}^{q}2^{2q+1-2j} \\ &= n+2(n-3)\sum_{i=0}^{q-1}4^i = n+2(n-3)\frac{4^q-1}{3} \\ n_a &= n+(n-3)\frac{2^n-2}{3} \end{aligned}$$

Therefore, after some calculation we find that

$$n_a - n_l = 2+\frac{n+3}{3}\left[1-2^{n-1}\right]. \tag{4.5}$$

For $n \geq 3$ the term $(n+3)/3$ is ≥ 2. Hence, observing that its coefficient $1-2^{n-1}$ is negative, the difference (4.5) satisfies

$$n_a - n_l \leq 2 + 2(1-2^{n-1}) = 4 - 2^n < 0$$

which leads again to a contradiction.

∎

Theorem 4.2 excludes the cases $n \leq 2$, but it is clear that a 2-cube can be transformed into a 1-level tree by removing any of its 4 nodes. We show in the next proposition that we can imbed a binary tree of height lower than n in n–cube.

Proposition 4.2. *An n-cube contains a binary tree of height* $\lceil \frac{n+1}{2} \rceil$.

Proof. Let us build the tree recursively, starting from node 0^n as its root, where as before the powers refer to concatenation. We can take the following nodes as the children of the root

$$01\ 0^{n-2};10\ 0^{n-2}$$

Generally, given a node of the form $A\ 0^{n-2i}$ at level i we take as its children the nodes

$$A\ 01\ 0^{n-2i-2} \equiv A'\ 0^{n-2(i+1)} \quad \text{and} \quad A\ 10\ 0^{n-2i-2} \equiv A''\ 0^{n-2(i+1)}.$$

This process can be pursued until $n - 2i \leq 1$ or $i \geq (n-1)/2$, i.e., until $i = \lceil \frac{n-1}{2} \rceil$ which gives a total of $i + 1 = \lceil \frac{n+1}{2} \rceil$ levels.

∎

We do not know whether Proposition 4.2 is optimal, i.e., whether one can imbed a binary tree of height $> \lceil \frac{n+1}{2} \rceil$ in an n-cube. However, a question which seems more relevant for practical purposes is whether or not it is possible to map a structure which resembles a binary tree into an n–cube.

An interesting structure of this form can be deduced from the mapping used in [4] in the context of multigrid algorithms. In the concurrent multigrid method, one has a number of levels of one-dimensional meshes consisting of 2^n grid-points for the level 0, 2^{n-1} for level 1, and so on. When passing from a fine grid to a coarse grid we simply reproduce every other point of the fine grid. Thus, the physical points of the finest grid are represented several times in order to allow for simultaneous work at all levels. Then in order to mimimize communication costs, the problem is to assign these $2^{n+1} - 1$ points into an $(n+1)$-cube in such a way that the distance between successive points of the same level is one while the distance between two points that are on different levels but correspond to the same physical point is small.

Here, the role of the binary reflected Gray code described in Section 4.1, is crucial and the mapping given in [4] makes extensive use of it. Let us number the grid points of the l^{th} level, $l = 0, 1, .., n-1$ by $x^{(l)}_{2^l i}, i = 0, 1, \ldots 2^{n-l-1}$. Thus two points of different levels (different superscripts) have the same subscript when they correspond to the same physical point. If we denote by $g^{(l)}_i, i = 0, 1, \ldots 2^{n-l} - 1$ the $(n-l)$-bit binary reflected Gray code then the mapping

$$x^{(l)}_{2^l i} \longrightarrow g^{(l)}_i 10^l$$

is such that the distance between neighboring points on the same level is one and the distance between the grid point $x^{(l)}_j$ of level number l and $x^{(l-1)}_j$ of level number $(l-1)$ is at most two, c.f., [4]. In fact it is easy to show that the point $x^{(l)}_i$ is directly connected to either $x^{(l-1)}_j$ or $x^{(l-1)}_{j+1}$ and

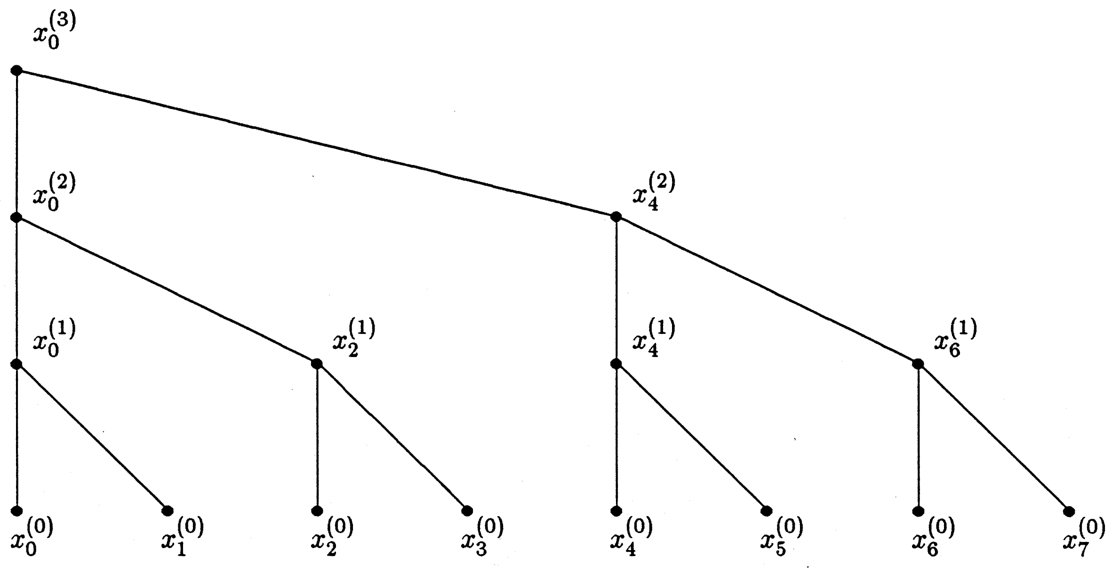


Figure 6: Binary tree-like mapping

these two points are always directly connected to each other. Hence the structure of the graph of all mesh points at all levels resembles that of a binary tree: $x_j^{(l)}$ can be considered as the father of $x_j^{(l-1)}$ and $x_{j+1}^{(l-1)}$, if we allow for edges to be either a real edge of a path of length 2 of the hypercube. This structure is illustrated in Figure 6 where the lines show the pseudo edges which in fact are paths of length either one (direct connection) or two (connection through via one neighbor). This means that we can map a tree structure into the hypercube so that the distance between any two adjacent points of the tree are at distance at most two from each other in the hypercube. The mapping is obtained by simply looking at the node labels to which each of the grid points is assigned. It can be briefly described for an $(n+1)$-cube by scanning its nodes of level number $l, l = 0, 1, \dots, n$, from left to right, the first level ($l = 0$) corresponding to the leaves: $g_i^{(l)}10^l, i = 0, \dots 2^{n-l} - 1$. The resulting pseudo binary-tree mapping is shown in Figure 7, for a 4–cube.

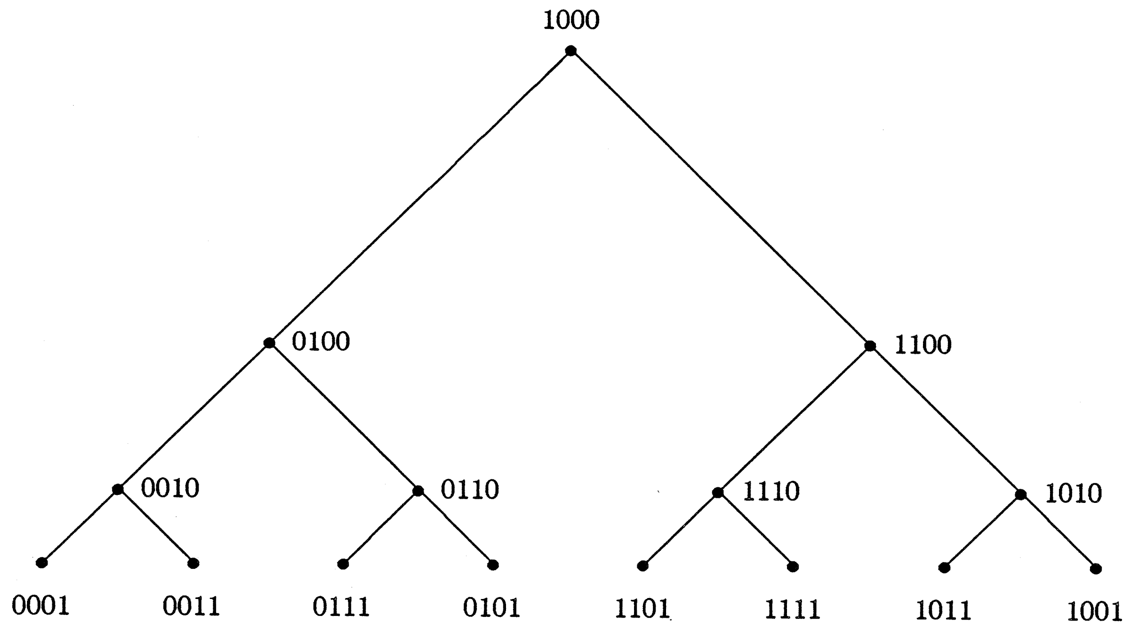


Figure 7: Pseudo imbedded binary tree: the edges represent either an edge of the hypercube or a path of length two

References

[1] L.M. Adams, *Iterative algorithms for large sparse linear systems on parallel computers,* Ph.D. Thesis, University of Virginia, Applied Mathematics, 1982. Also available as NASA Contractor Report# 166027.

[2] L. N. Bhuyan, D.P. Agrawal, *Generalized Hypercube and Hyperbus structures for a computer network,* IEEE Trans. Comp., C-33 (1984), pp. 323–333.

[3] S.A. Browning, *The tree machine: A highly concurrent computing environment,* Technical Report TR-3760, California Institute of Technology, Computer Science, 1980.

[4] T.F. Chan, Y. Saad, *Multigrid Algorithms on the Hypercube multiprocessor,* Technical Report 368, Computer Science Dept., Yale University, 1985.

[5] A. Gottlieb, R. Grishman, C.P. Kruskal, K.P. Mc Auliffe, L. Rudolph, M. Snir, *The NYU Ultracomputer - Designing an MIMD shared memory parallel computer,* IEEE Trans. Comp., /C-32 (2) (1983), pp. 175–189.

[6] E.M. Reingold, J. Nievergelt, N. Deo, *Combinatorial Algorithms,* Prentice Hall, New-York, 1977.

[7] C.L. Seitz, *The Cosmic Cube,* CACM, 28 (Jan. 1985), pp. 22–33.