Abstract

The purpose of embedding (or mapping) is to minimize the communication overhead of parallel algorithms. It appears unlikely that an efficient exact technique of embedding a general guest graph in a general host graph with a minimum-dilation will ever be found. The research in this area is concentrating on discovering efficient heuristic that find solutions in most cases.

This article is a survey of techniques for embedding common guest graphs - such as linear arrays, trees, meshes, etc. into hypercubes. The reason of hypercube as a host graph is that hypercube topology have been considered as parallel architecture due to its powerful interconnection network. We examine the hypercube and embedding problem from the graph theory point of view. Among other things we propose a mathematical framework of the hypercube as an interconnection network and look into a theoretical characterization of embedding various topology into a hypercube.

1 Introduction

In parallel computation, interconnection networks are structured into various configurations such as trees, pyramids and mesh arrays. These configurations can be represented as graphs. If the structure of graphs and properties are used effectively, the computation and communication speeds can be improved. This survey paper studies the mapping computations of different networks configurations such as arrays, trees, and meshes of trees onto hypercubes.

The hypercube is one of the most adaptable and efficient networks yet discovered for parallel computation. The hypercube network is compatible for both special-purpose and general-purpose computation. A hypercube of degree, d, has exactly, d, neighbors. The distance between any two nodes is less than or equal to d. Both the diameter of the hypercube and the degree of the nodes grow very slowly (logarithmically) with respect to the number of nodes in the hypercube. The hypercube configuration can transmit a large number of data rapidly. This makes the hypercube a more useful and effective intercon-
nection network than networks such as array and trees.

The paper is divided into five sections. The paper start with a discussion of the hypercube’s most important properties in section 2. Among other things we show the techniques for embedding arrays, trees, and meshes of trees on a hypercube in section 3. Section 4 discuss the communication issues and summary and conclusion in section 5.

2 Preliminaries

This section presents basic definitions used in this survey paper.

2.1 Boolean Hypercube And Graycodes

The d-dimensional hypercube, $A_d$, consists of $N = 2^d$ nodes with distinct d-bit address and $d2^{d-1}$ edges. There is a directed edge (communication link) $(u,v)$ if and only if the addresses of $u$ and $v$ differ in exactly one bit position. An edge between two nodes that differ in the $i^{th}$ bit is said to be the $i^{th}$ dimension.

The binary reflected graycode transition sequence, $Q_k'$, is defined as $Q'_1$ and $Q'_{i+1} = Q'_i \circ i \circ Q'_i$, $0 \leq i < k$. Further, define $Q_k = Q'_k \circ k - 1$ and let $Q_k(j)$ denote the $j^{th}$ element of, $0 \leq jj < 2^k$, where, $\circ$ represents sequence concatenation.

2.2 Node Parity

A hypercube node has even parity if its corresponding binary string has an even number of ones. Otherwise, the node is said to have odd parity. Since the N-node hypercube has an edge link with different parity, it has $\frac{N}{2}$ odd nodes. For example, the 8-node hypercube, with even parity nodes high lighted

Figure 1: 8-node Hypercube

2.3 Graph Embedding

An embedding of a guest graph $G = (V,E)$, onto a host graph $H = (W,E)$, is a one-to-one mapping $\eta : V \rightarrow W$ in conjunction with a map $\mu$ which assigns each edge $(u,v) \in E$ to path in $H$ from $\eta(u)$ to $\eta(v)$. When $|V| > |W|$ we allow many-to-one mappings, but limit the mapping so that each host vertex is the image of no more than $\frac{|V|}{|W|}$ guest vertices. The quantity $\frac{|V|}{|W|}$ is called the load of the embedding. Four cost metrics are used to evaluate the quality of an embedding:

1. Dilation is defined as the maximum distance in host graph, $H$, between the images of two adjacent nodes of guest graph, $G$. The larger the dilation, the longer is the maximum latency among communicating node-pairs. Thus, dilation is a measure of the communication performance of the mapping. For example, the dilation of an edge $e \in E$ is the length of the path $\mu(e)$ and the dilation of an embedding is the maximum dilation of any edge in guest graph, $G$.

2. Expansion is the ratio of the total number of nodes in host graph, $H$, to the total number of nodes in guest graph; $G$. Expansion is a measure of processor utilization for a given embedding.

3. Congestion is defined as the maximum
number of edges in guest graph, G, that share the same edge of host graph, H. Clearly, a large value of congestion indicates a high degree of sharing among communication paths in the given embedding, indicating long queuing delays at nodes and possible congestion. For example, the congestion of an edge $f \in F$ equals the number of edges in G whose image contain f. The congestion of an embedding is the maximum congestion of any edge in H.

4. Maximum load is defined as the maximum number of nodes of guest graph, G, that are mapped to the same node in host graph, H. Maximum load indicates the maximum number of tasks assigned to a processor in the system, and thus measures the processing time.

2.4 Cost Of Embedding

The one-packet cost of an embedding of guest graph, G, into host graph, H, is the number of time units necessary for host graph, H, to complete one phase of guest graph, G, when each message contains one packet.

The p-packet cost of an embedding of G into H is the number of time units necessary for H to complete one phase of G in which each message contains p packets.

2.5 Copy And Path Embeddings

A k-copy embedding of guest graph, G into host graph, H, is a collection of k one-to-one embeddings of G into H. A width-w embedding of guest graph, G, into host graph, H, is a one-to-one embedding in which each edge of G is mapped to w edge-disjoint paths in H.

2.6 Cross Product

The cross-product of two graphs $G = (V, E)$ and $H = (W, F)$ is denoted $G \times H$ and consists of vertex set $V \times W = \{< v, w > | v \in V, w \in W\}$ and set $\{(< v, w_1 >, < v, w_2 >) | v \in V, w_1, w_2 \in W\} \cup \{(< v_1, w >, < v_2, w >) | v_1, v_2 \in V, w \in W\}$. By analogy with multiplication, the graphs G and H are referred to as factors of $G \times H$. The cross product of the length L path and the length W path is the $L \times W$ grid. Similarly, the cross product $Q_m \times Q_m$ is equal to $Q_{n+m}$.

3 Mapping Other Geometries Into Hypercube

Given some graph $G = (v, E)$ having no more than $2^n$ vertices and the problem is to assign the vertices of the graph into the nodes of the d-dimensional hypercube so that every adjacent vertex of the graph belong to neighboring nodes of the n-cube.

There are two reasons why such mappings are important.

1. Algorithms may be developed for some other architecture. One would like to implement the same algorithm on hypercube. Then the original architecture can be mapped into the hypercube.

2. A given problem have a well-defined structure(architecture) which leads to a particular patterns of communication mapping the structure to other structure may result in substantial saving in communication time.
3.1 Embedding of Array in Hypercube

3.1.1 Linear Dimensional Array

The N-node hypercube contain high-dimensional arrays and arrays with wraparound. For example, the embedding of a $4 \times 4$ array in a 16-node hypercube is shown in figure.

Figure 2: Embedding of a $4 \times 4$ array in a 16-node Hypercube

In order to show that hypercube contains a linear array as subgraph we show that the hypercube is Hamiltonian. The result can be extended to higher dimensional arrays.

**Theorem 1** The N-node hypercube contains N-node linear array (with wraparound) as a sub-graph for $N \geq 4$.

**Proof.** The proof is by induction on N. The base case $N = 4$ is true by inspection. Suppose that result is true for $\frac{N}{2}$. Then partition the hypercube into two $\frac{N}{2}$ sub-hypercubes.

By induction, it is possible to construct similar Hamiltonian cycles in each subcube. By symmetry, we can construct the full Hamiltonian cycle by replacing corresponding two edges as shown in figure.

3.1.2 Higher-Dimensional Array

In order to show higher dimensional arrays is a subgraph of a hypercube.

Use the following mathematical properties.

- The higher-dimensional array is a cross product of linear arrays.
- The hypercube is a cross product of smaller cube.

Formally, a graph $G = (V,E)$ is the cross product of $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2), \ldots, (V_k, E_k)$, i.e.,

$$G = G_1 \times G_2 \times \cdots \times G_k$$

if $V = \{(v_1, v_2, \ldots, v_k) | v_i \in V_i \text{ for } 1 \leq i \leq k\}$

and

$$E = \{(u_1, u_2, \ldots, u_k), (v_1, v_2, \ldots, v_k) \mid \exists j \text{ such that } (u_j, v_j) \in E_j \text{ and } u_i = v_i \text{ for } i \neq j\}$$

In general, $M_1 \times M_2 \times \cdots \times M_k$ arrays is the cross product of linear arrays of size $M_1, M_2, \ldots, M_k$. This is because two nodes $(u_1, u_2, \ldots, u_k)$ and $(v_1, v_2, \ldots, v_k)$ of the arrays are linked by an edge (communication link) if and only if $|u_j - v_j| = 1$ for some $j$ where $1 \leq j \leq k$ and $u_i = v_i$ for all $i \neq j$. This
means that a hypercube can be represented as a cross product of subhypercubes in many different ways. In particular, the following theorem characterizes a broad and useful class of crossproduct representations for a hypercube.

**Theorem 2** The d-dimensional hypercube $H_d$ can be expressed as the crossproduct $H_d = H_{d_1} \times H_{d_2} \times \cdots \times H_{d_k}$ for any $k \geq 1$. Where $H_{d_i}$ denotes the $d_i$-dimensional hypercube for $1 \leq i \leq k$.

*Proof. The d-dimensional hypercube $H_d$ is a $2 \times 2 \times \cdots \times 2$ array. Since $H_{d_i}$ is the cross product of $d_i$ linear array,

$$H_d = H_1 \times H_1 \times \cdots \times H_1 \times H_1$$

$$= H_{d_1} \times H_{d_2} \times \cdots \times H_{d_k}$$

$$= H_{d_1} \times \cdots \times H_{d_k}$$

as claimed.

From the theorem 1 and 2, we know a multidimensional array is a cross product of linear arrays and a hypercube is a cross product of smaller hypercubes.

From the above analysis we know that the following facts:

- A multidimensional array is the cross product of linear arrays.
- A hypercube is a cross product of smaller hypercubes.
- A linear array is a subgraph of a hypercube.

**Theorem 3** If $G_1 = G_1 \times G_2 \times \cdots \times G_k$ and $H_1 = H_1 \times H_2 \times \cdots \times H_k$ for some $k \geq 1$ and $G_i$ is a subgraph of $H_i$ for $1 \leq i \leq k$, then $G$ is a subgraph of $H$.

*Proof. Construct one-to-one mapping of the nodes of $G$ to the nodes of $H$ that maps edges to edges. This equivalent to showing that $G$ is a subgraph of $H$. For each $i$ such that $1 \leq i \leq k$, let $\delta_i$ be a map of the nodes of $G_i$ to the nodes of $H_i$ that preserves edges. Given any node $v = (v_1, v_2, \ldots, v_k)$ of $G$, define $\delta : G \rightarrow H$ by $\delta(v) = (\delta_1(v_1), \delta_2(v_2), \ldots, \delta_k(v_k))$ To see that $\delta$ preserves edges, we need to show if $(u, v)$ is an edge of $G$, then $\delta(u), \delta(v))$ is an edge of $H$. This easy to do since if $(u, v)$ is an edge of $G$, then there is a $j$ such that $(u_j, v_j)$ is an edge of $G_j$ and such that $(u_i = v_i)$ for all that $i \neq j$. Hence, for the same $j$, $(\delta_j(u_j), \delta_j(v_j))$, is an edge of $G_j'$ and $(\delta_i(u_i) = \delta_i(v_i))$ for all $i \neq j$. Thus $(\delta(u), \delta(v))$ is an edge of $H$. We can now infer from the facts and theorem 3 that a multidimensional array is a subgraph of a hypercube i.e., a $2^{d_1} \times 2^{d_2} \times \cdots \times 2^{d_k}$ array is a subgraph of the $2^d$-node hypercube where $d = d_1 + d_2 + \cdots + d_k$. In particular, this means that any $2^d$-node array of any dimension is a subgraph of the $2^d$-node hypercube. More, generally, we conclude that an $M_1 \times M_2 \times \cdots \times M_k$ array is contained in an $N$-node hypercube, where

$$N = 2^{[lgM_1]+[lgM_2]+\cdots+[lgM_k]}$$

Since any array with wraparound edges is the cross product of linear arrays with wraparound, therefore, we can extend the preceding results to array with wraparound.

### 3.2 Embedding of Binary Tree in Hypercube

Despite of the fact that hypercube has an inherent binary structure, on (N-1)-node complete binary tree cannot be embedded in a N-node hypercube with a dilation cost of 1 and an expansion lost less than 2.
Theorem 4 A complete binary tree (CBT) of height \( d > 2 \) cannot be embedded in a hypercube of degree \( d \) less than or equal to \( d \) such that adjacency is preserved.

Proof. A hypercube of degree less than height \( d \), has fewer nodes that \( CBT_d \); therefore, \( CBT_d \) cannot be embedded. Consider a mapping \( f \) of nodes \( CBT_d \) into a hypercube, \( Q_d \), of degree \( d \). Let the nodes of \( Q_d \) are labeled with binary numbers between 0 and \( 2^d \). without loss of generality, we may assume that the root of \( CBT_d \) at level 0 is mapped onto node 0 in \( Q_d \). Since level 0 node, 1, has no 1’s in its label, the following are true.

1. 1 (level \( i \)) node has no more than \( i \) 1’s in its label.
2. 1 (odd level) node has an odd number of 1’s in its label.
3. 1 (even level) node has an even number of 1’s in its label.

Case 1: \( d \) is even (even parity).

The number of binary strings of of length \( d \) with an odd numbers of ones is
\[
\sum_{i=0}^{d} \binom{d}{i} = 2^{d-1} - 1.
\]

The number of odd level nodes in \( CBT_d \) is \( 2 + 2^3 + 2^5 + \cdots + 2^{d-1} = \frac{2}{3}(2^d - 1) \). But \( 2^{d-1} < \frac{2}{3}(2^d - 1) \) if and only if \( d > 2 \).

This proof that if height \( d \), is even and height \( d \) \( \leq 2 \), there are not enough node in hypercube, \( Q_d \), with an odd number of 1’s in their labels to the images of the odd level nodes of complete binary tree, \( CBT_d \).

Case 2: \( d \) is odd (odd parity).

The number of binary strings of length with an even number of 1’s is \( \sum_{i=0}^{even} \binom{d}{i} = 2^{d-1} \). The number of even level nodes in \( CBT_d \) is \( 2^0 + 2^2 + 2^4 + \cdots + 2^{d-1} = \frac{(2^d-1)}{3} \). But \( 2^{d-1} < \frac{(2^d-1)}{3} \) if and only if \( d > 1 \). This shows that if \( d \) is odd and \( d \leq 1 \), then there are not enough nodes in hypercube, \( Q_d \), with an even number of 1’s in their labels to be the images of the even label nodes of complete binary tree, \( CBT_d \).

There is no adjacency-preserving embedding of complete binary tree, \( T_d \), into hypercube, \( Q_d \), for \( d > 2 \). However, next theorem will show that if we allow a hypercube, with twice as many nodes, as the host graph, then there is an adjacency-preserving embedding of complete binary tree, \( CBT_d \), into it. That is, there is an embedding with dilation cost of 1 and an expansion of 2.

Theorem 5 For \( d > 0 \), a complete binary tree, \( CBT_d \), can be embedded in a hypercube \( Q_{d+1} \) of degree \( d + 1 \), such that the adjacencies of nodes \( CBT_d \) are preserved.

Proof. The Proof is by Induction. Following figure illustrates the mapping of \( CBT_1 \) of degree 1 \( CBT_2 \) of degree 2, and \( CBT_3 \) of degree 3 onto hypercubes hypercubes \( Q_1 \), \( Q_2 \) and \( Q_3 \) respectively with adjacency preserved.

Let \( f_i : CBT_i \rightarrow Q_{i+1} \) be a mapping of complete binary tree of higher \( i \), \( CBT_i \), into hypercube of degree \( i + 1 \), \( Q_{i+1} \). Suppose \( f_{d-1} : Q_d \rightarrow Q_d \) preserves adjacency and \( (Q_d, f_{d-1}) \) has the following free-free neighbor "property: \( R = f_{d-1}(\text{root of } CBT_{d-1}) \) has a free neighbor A and A has a free neighbor \( B \neq R \) i.e., \( \{A,B\} \subseteq \{f_{d-1}(N) \mid N \text{ a node of } CBT_{d-1}\} \). Note that \( (Q_4, t_3) \) has the "free-free neighbor" property.

\[\binom{d}{i} = \sum_{j=0}^{d} \binom{j}{i}2^{j-1}\]
The left structure of a complete binary tree of height \( d \), \( CBT_d \), can be mapped by \( f_{d-1} \) in \( OQ_d \) of \( Q_{d+1} \), such that \( OL = f_{d-1} \) (root) has a free neighbor OB. The right subtree can also be embedded in a degree \( d \) hypercube \( Q_d \), using \( t'_{d-1} \) such that \( R' = f_{d-1}(\text{root of right subtree}) \) has a free neighbor \( A' \) which has a free neighbor \( B' \) in \( Q_d \). We can now apply transformation \( T: Q'_d \rightarrow 1Q_d \) such that \( T(R') \) is the neighbor of OA in \( 1C_d \) and \( T(A') \) is the neighbor of OB in \( 1Q_d \). This procedure is similar as forming a degree \( d + 1 \) hypercube from two degree \( d \) hypercubes. Now the mapping \( t_d \) of \( CBT_d \) onto \( Q_{d+1} \) defined as: \( f_d(\text{root of } CBT_d) = OA, f_d \) such that left subtree = \( f_{d-1} \), \( f_d \) such that right subtree = \( T(f'_{d-1}) \). Distances between OA and the image of the left and right children of the root are both 1. More over, the free-free neighbor property is satisfied by \((Q_{d+1}, f_d)\) since OA has a free neighbor OB and OB has a free neighbor \( T(A') \).

Therefore, by Induction hypothesis, a complete binary tree of height \( d \geq 0 \) can be embedded in a hypercube of degree \( d + 1 \) with adjacency preserved. Conceptually is a process of building up from the smaller subtrees. The free-tree neighbor property ensures that a free neighbor is available in the embedding of a taller tree.

The above theorem show the technique to embed a complete binary tree in a hypercube. If a binary tree is not complete, add imaginary nodes and then embedded it in a hypercube using theorem.

### 3.3 Embedding of Meshes of Trees in Hypercube

Of all the networks that are contained in the hypercube, the most important are meshes of trees. By efficiently embed meshes of trees in a hypercube, we show that the hypercube is at least as powerful as arrays, trees, and meshes of trees combined.

**Theorem 6** \( MOT(2n, 2n) \) can be embedded in hypercube \( Q2n+2 \) with dilation cost is equal to one and optimum expansion cost.

Before proving the theorem we look at the embedding of a simpler structure, called back-to-back trees (BB). Back-to-back trees is essentially a combination of two equal sized complete binary trees which share the leaf nodes. A BB tree with eight leaves is shown below.

Here our goal is to study the embedding of the architecture a a preliminary study for the proof of the above theorem 3.
Theorem 7  Back-to-back trees can be embedded in the hypercube with unit dilation cost and optimum expansion cost.

Proof: A BB tree with $2^n$ leaves contains $3 \times 2^n - 2$ vertices. Therefore, $Q_{n+2}$ is the smallest hypercube to accommodate that structure. We construct BB trees with tails attached to each of the two roots. The shape of the tree depends on whether $n = 2k$ or $n = 2k + 1$ for some integer $k$.

Consider the case $n = 2k$. Figure 6 shows the embedding of a four-leaf BB tree with tails in $Q_4$. Generally, a $2^{2k}$-leaf BB tree with tails is available in $Q_{2k+2}$. A BB tree with $2^{2k+1}$ leaves can be constructed in $Q_{2k+3}$ by carrying out the following steps:

1. Copy the initials BB tree into $Q_{2k+2}$ and connect the two hypercubes.

2. Apply the translation polynomial $P_n(v) = (1 + v_i)2^i$ on every vertex $v$ of the new copy.

3. Add the edges.

After this construction the new tails are obtained by adding the edges for the upper root. A similar construction applies for the lower root. This shape facilitates a simple construction for the next iteration so that the original shape of the tails are recovered. We can get a sequence representing the successive dimensions along which the right BB tree is translated at each step of construction. There are, of course, several other possibilities. Deriving such a sequence yields an algorithm for constructing the BB tree in the hypercube described as:

1. Make a copy of original BB tree.

2. Translate the new copy along the next dimension in the sequence.

3. Connect the roots.

Proof of Theorem 6  Given this background we can describe the construction method for the two dimensional MOT. The basic idea is again to construct MOT with tails at every root. The general construction uses a two-phase procedure, where the first phase extends the row trees to obtain MOT($2^n, 2^{n+1}$) from MOT($2^n, 2^n$), and the second phase extends the column tree to obtain MOT($2^{n+1}, 2^{n+1}$). Since the smallest hypercube with enough vertices contains $2^{2n+2}$ vertices, optimal expansion cost embedding requires a $(2n + 2)$-dimensional hypercube for MOT($2^n, 2^n$). The embedding of MOT($2^n, 2^n$) in $Q_8$. Note the following properties of this embedding:

1. For every row tree the tail structure is a cycle of four vertices containing the same pair of dimensions $i$ and $j$.

2. For every column tree the tail structure is a cycle of four vertices containing the same pair of dimensions $k$ and $l$.

We describe a recursive construction method which relies on the above two properties and preserves them at every step. First consider obtaining MOT($2^n, 2^{n+1}$) from...
MOT(2^n, 2^n) there MOT(2^n, 2^n) contains m row trees and m column trees, where m = 2^n. The following are the steps of construction.

1. Copy initial graph into Q_{2n+n} and connect the two hypercubes. Observe that the cycle at the tail of a row tree construction a three-dimensional hypercube with the corresponding cycle in its image at dimension i, j, 2n + 3. This is true for every row tree.

2. Apply the translation polynomial \( P(v) = (1 + v_i)2^i \) on every vertex v of the new copy. This transformation preserves the shape of the graph.

3. Add the edges and the new row roots are computed for all i.

After this construction the new tails are obtained by adding the edges. Note that the shape of the tails is now different. This shape facilitates a simple construction, similar to that in theorem 7, for the original tail shape will be recovered. The extension method for the column tree is similar therefore it is omitted to avoid repetition.

In this section we developed embedding methods for one-, two-, and three-dimensional mesh of trees. Extension to multidimensional mesh of trees can be achieved in a similar way. This embedding has considerable practical importance since mesh of trees enable extremely high-speed parallel computation. An interesting property common to all of the embedding methods presented here is that the methods yield simple algorithms even though their proofs were some what elaborate.

4 Communication Issues

A large amount of processing time is spent in routing data among processor. Therefore, routing algorithms, techniques, and models are primary importance for an efficient use of parallel computer. Several machine-related features, specific protocol and the setting of the destination addresses have to be taken account in order to design fast routing algorithms. In parallel computing, packet routing has been implemented to support interprocessor communication. If a processor sends a message to a processor other than its neighbor. Under the Store-and-Forward protocol, every processor in the path taken by the message first stores the message then forwards it to the next processor in the path. Experimentation showed that the cost of a sending-to-neighbor operation could be modeled as a start-up time \( \beta \) plus the time spent by the actual sending of the messages, which depends on the size of the message (L) and on the inverse of the link’s bandwidth (\( \tau \)). So, the time for such an operation could be expressed as \( t = \beta + L\tau \), known as the linear model. Another important parameter for routing problems is the concurrency of communication links, i.e., the number of links that can be used concurrently by each processor. Communication allow by algorithms with only one neighbor are said to be the 1-port model. If communication with all the d neighbors is allowed, then it said to be the d-port model.

5 Conclusion

The hypercube is a good host graph for the embedding of networks of processors because of its low degree and low diameter. Graphs such as trees and arrays can be embedded into
a hypercube with small costs. The design of
the structure s of these graphs. In general,
there is a trade-off between the dilation cost
the expansion cost. This paper has shown
that there is no embedding of a complete bi-
nary tree into hypercube with dilation cost of
1 and expansion cost less than 2. But there is
an embedding with dilation cost 1 and expan-
sion cost approximately 2. For embedding of
general graphs, if the size of the hypercube
is minimal, then the cost of embedding may
be as large as log n where n is the number of
nodes in the graph.

We started with a discussion of the hyper-
cube’s most important properties. Among
other things, we showed how array, trees and
meshes of trees can be embedded on a hy-
percube. As a consequence, we immediately
understood one of the reason why the hyper-
cube is so useful.

References

ing Problem,” IEEE Trans. Com-

Network into Hypercubes,” J.Parallel
and Distributed Computing, vol.2,

[3] Sajay R. Deshpande and Roy M. Jen-
evein, “Scalability of a Binary Tree on
Parallel Processing, pp.661-668, 1986.

[4] Ching-Tien Ho and S. Lennart
Johnsson, “On Embedding of Arbi-
trary Meshes in Boolean Cubes
with Expansion Two Dilation Two,”
Proc. Int’l Conf. Parallel Processing,

[5] Tony F. Chan and Youcef Saad,
“Multigrid Algorithms on the Hyper-
cube Multiprocessor,” IEEE Trans.
Comput., vol.C-35, no.11, pp. 969-
977.

among Meshes and Tori,” J. Paral-
el and Distributed Computing, vol.18,

[7] Rami G. Methem and Ghil-Young
Hwang, “Embedding Rectangular
Grids into Square Grids with Dilation
Two,” IEEE Trans. Comput., vol.39,

[8] Youcef Saad and Martin H. Schultz,
“Topological Properties of Hyper-

[9] Laximi N. Bhuyan and Dharma P.
Agarawal, “Generalized Hypercube
and Hyperbus Structures for a Com-
puter Network,” IEEE Trans. Com-

Embedding Rectangular Grids in Hy-

Connecting Networks and Telephone
Traffic,” Academic Press, New York,
1965.

[12] Ming-Syan Chen and Kang G. Shin,
“Processor Allocation in an N-cube
Multiprocessor Using Gray Codes,”
IEEE Trans. Comput., pp. 1396-1407,


