The Journal of Supercomputing, 8, 263–294 (1994)

Incomplete Hypercubes: Algorithms and Embeddings

ALFRED J. BOALS, AJAY K. GUPTA, AND NAVEED A. SHERWANI*
Department of Computer Science, Western Michigan University, Kalamazoo, MI 49008-5021

(Received June 1992; final version accepted June 1994.)

Abstract. The hypercube, though a popular and versatile architecture, has a major drawback in that its size must be a power of two. In order to alleviate this drawback, Katseff [1988] defined the *incomplete hypercube*, which allows a hypercube-like architecture to be defined for any number of nodes. In this paper we generalize this definition and introduce the name *composite* hypercube. The main result of our work shows that these incomplete architectures can be used effectively and without the size penalty. In particular, we show how to efficiently implement Fully Normal Algorithms on composite hypercubes. Development of these types of algorithms on composite hypercubes allows us to efficiently execute several algorithms concurrently on a complete hypercube. We also show that many host architectures, such as binary trees, arrays and butterflies, can be optimally embedded into composite hypercubes. These results imply that algorithms originally designed for any such host can be optimally mapped to composite hypercubes. Finally, we show that composite hypercubes exhibit many graph theoretic properties that are common with complete hypercubes. We also present results on efficient representations of composite hypercubes within a complete hypercube. These results are crucial in task allocation and job scheduling problems.

Keywords. Hypercube, algorithms, incomplete hypercubes, graph embeddings.

1. Introduction

The hypercube has emerged as a popular topology for parallel machines, and several hypercube machines (e.g., Intel iPSC and NCUBE) are commercially available. The popularity of this architecture is due to its regular structure and its rich interconnection topology. These properties allow for the development of simple and efficient algorithms for node-to-node communication and broadcasting, which are the building blocks for other parallel algorithms. However, the hypercube requires the number of nodes to be a power of two. In addition, to take advantage of the symmetry in the hypercube topology, it is usually assumed that the hypercube is *complete*. In practice, due to budgetary constraints or node failures, it may not always be possible to have a complete architecture. This frequently happens while upgrading an existing architecture to the next dimension. For example, if a five-dimensional 32-node hypercube is upgraded, an additional 32 nodes are needed before any algorithm requiring nodes in the range of 33 to 64 may be run.

In order to address this unrealistic node requirement, two architectures have been presented [Chen and Tzeng 1989; Katseff 1988; Tien and Yang 1991; Tzeng et al. 1990]. Katseff [1988] described an incomplete hypercube $IH(m)$ consisting of nodes labeled 0, 1, ..., $m - 1$

*This research was supported in part by the National Science Foundation under grant USE-90-52346. A preliminary version of this work appeared in the *5th International Parallel Processing Symnposium*, May 1991.

of an n-node complete hypercube Q_n along with their interconnections in Q_n. He showed that routing and broadcasting algorithms may be performed on $IH(m)$ in a manner analogous to that of a complete hypercube. Tzeng et al. [1990] described a more restricted definition of $IH(m)$ by considering only those situations when $m = 2^{d-1} + 2^{d-2}$ if $n = 2^d$; hence this architecture may be viewed as a composition of two complete hypercubes $Q_{n/2}$ and $Q_{n/4}$.

We generalize the notion of a composition of several smaller sized hypercubes to obtain an arbitrary-sized architecture, which we will call a *composite* hypercube $CH(m)$. A composite hypercube can be defined for any arbitrary m and can be easily upgraded to a complete hypercube. We show that this concept allows the algorithms to work efficiently even if the hypercubes are incomplete. Recall that a d-dimensional hypercube Q_d has 2^d nodes and every node in Q_d is labeled as $b_{d-1}b_{d-2} \ldots b_0$ where $b_s \in \{0, 1\}$ for $0 \leq s \leq d - 1$. A node with label $b_{d-1} \ldots b_0$ is connected to d nodes having labels $b_{d-1} \ldots b_{s+1}\bar{b}_s b_{s-1} \ldots b_0$, for $0 \leq s \leq d - 1$. Equivalently, if $n = 2^d$ then the nodes of Q_d can also be identified with integers 0, 1, ..., $(n - 1)$ so that if a pair of nodes i and j are connected, then $i - j = \pm 2^p$ for some $p \geq 0$. We proceed by giving a recursive definition of a composite set that we will use to define composite hypercubes.

Definition 1. *A set of nodes S in a hypercube Q_d is called a* composite set *if*

1) $|S| = 2^r$ *for some* $0 \leq r \leq d$ *or*
2) *there exists a positive integer* $k \leq d$ *such that* $2^{k-1} < |S| < 2^k$, *and a* k*-dimensional hypercube* Q_k *in* Q_d *containing* S *as a subset of its nodes with the property that* Q_k *contains two disjoint* $(k - 1)$*-dimensional hypercubes* Q^0_{k-1}, Q^1_{k-1} *such that the node set* V^0 *of* Q^0_{k-1} *is a subset of* S *and* $S - V^0$ *is a composite set in* Q^1_{k-1}.

Note that in the above definition of a composite set the hypercube Q_k is unique, but the hypercubes Q^0_{k-1}, Q^1_{k-1} are not necessarily unique. However, as will be shown by Lemma 1 the hypercube Q_d can be relabeled so that the vertices of a composite set have consecutive labels.

A *composite hypercube* is defined to be a subgraph of a hypercube that is induced by some composite set S. (Note that an induced graph on a set S of nodes in a graph G is the graph whose node set is S and whose edge set consists of those edges in G having both ends in S.) Figure 1 shows an example of a composite hypercube with 13 nodes. Note that composite hypercubes are a generalization of hypercubes as they are defined for any number of nodes. It is easy to see that the architecture defined by the schemes in [Katseff 1988], [Tien and Yang 1991], and [Tzeng et al. 1990] are specific cases of composite hypercubes.

In order to evaluate the effectiveness of a new architecture, one needs to at least investigate the underlying graph theoretic properties, graph embeddings and the programmability of the architecture. Several graph theoretic properties, such as diameter, connectivity and maximum degree, influence the maximum routing distance, stability under node failure and ease of fabrication due to a limited number of I/O ports per processor. Graph embeddings indicate various simulation capabilities of the architecture.

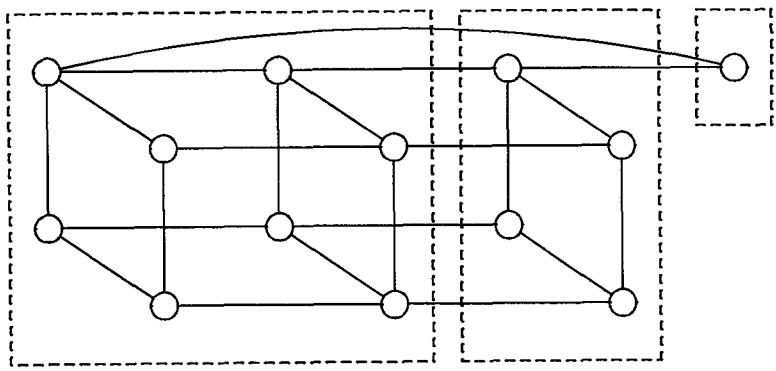


Figure 1. A composite hypercube of 13 nodes. Complete hypercubes forming the composite hypercube are shown in dashed boxes.

In this paper we first investigate several graph theoretic properties of composite hypercubes. We show that with respect to these properties composite hypercubes compare very well with hypercubes. We also describe an efficient algorithm to recognize a composite hypercube in a hypercube. This algorithm is critical in task allocation and scheduling algorithms using composite hypercubes, which are crucial for the development of efficient algorithms on the composite hypercubes.

We then investigate algorithmic properties of composite hypercubes. In particular, we show how to efficiently implement fully normal algorithms (FNA) on composite hypercubes. We also investigate the size/performance tradeoff for composite hypercubes. Empirical evidence is presented to show that the number of extra processors needed by a composite hypercube to effectively compete with a complete hypercube decreases logarithmically, as we increase the size of the hypercube.

We finally investigate efficient embeddings of various popular architectures into composite hypercubes. We present optimal embeddings of complete binary trees, meshes, butterflies and cube-connected cycles into composite hypercubes. These results, for the most part, are similar to the results on embeddings into complete hypercubes and hence show that composite hypercubes compete very well with complete hypercubes.

For some related results and previous work the reader may refer to [Chan and Lee 1993; Chen and Shin 1988; Chen and Tzeng 1989; Graham et al. 1993; Gordon and Stout 1988; Hastad et al. 1987, 1989; Kandlur and Shin 1988; Katseff 1988; Lee and Hayes 1988; Tien and Yang 1991; Tzeng 1990; Tzeng et al. 1990]. In [Hastad et al. 1987] we are shown how to reconfigure a hypercube in case some nodes are faulty. This is accomplished by the embedding of a complete hypercube into the one with faults. The problem of routing in the presence of faults has been investigated in [Hastad et al. 1989]. Since faults are considered with some probability, the algorithms are probabilistic. Several other papers consider routing, allocation and broadcasting problems on a faulty or "injured" hypercube [Chan and Lee 1993; Chen and Shin 1988; Gordon and Stout 1988; Kandlur and Shin 1988; Lee and Hayes 1988].

2. Properties and Recognition of Composite Hypercubes

In this section we investigate the graph properties of composite hypercubes, in particular, graph theoretic properties that are crucial in evaluating a parallel architecture. Graph theoretic properties of hypercubes have been extensively studied [Harary et al. 1988].

Let $CH(m)$ be a composite hypercube of m nodes, $2^{k-1} < m \leq 2^k$. Let $IH(m)$ be an incomplete hypercube as defined in [Katseff 1988]; that is, let V and E be the sets of nodes and edges in $IH(m)$. Then $IH(m)$ is an induced subgraph of a complete hypercube Q_k with $V = \{0, 1, \ldots, m-1\}$. Recall that the node set of Q_k is $\{0, 1, \ldots, 2^k - 1\}$. We next show that $CH(m)$ and $IH(m)$ are isomorphic. Because of this result, throughout the rest of this paper we refer to composite hypercubes as *incomplete hypercubes* and vice versa.

Lemma 1. *An m-node composite hypercube $CH(m)$ is isomorphic to an m-node incomplete hypercube $IH(m)$ for $m > 0$.*

Proof. Let $2^{d-1} < m \leq 2^d$ for some $d \geq 0$. Since $CH(m)$ and $IH(m)$ both are subgraphs of a complete hypercube Q_d, it is sufficient to show that the nodes of $CH(m)$ can be labeled (and relabeled) so that they are sequentially numbered from 0 to $m - 1$. Note that by the definition of composite hypercubes, $CH(m)$ consists of several smaller complete hypercubes of dimensions $d_1, d_2, \ldots, d_k$, where, $m = 2^{d_1} + 2^{d_2} + \ldots + 2^{d_k}$ and $d_1 > d_2 > \ldots > d_k$.

We prove the lemma by using induction on k. For the base case, note that $m = 2^{d_k}$; then clearly $CH(m)$ is isomorphic to $IH(m)$. Now let us assume by induction hypothesis that nodes of $CH(m - 2^{d_1})$ can be labeled from 0 to $m - 2^{d_1} - 1$. We need to show that nodes of $CH(m)$ can be labeled from 0 to $m - 1$ to form an $IH(m)$. In particular, we need to label the nodes of the d_1-dimensional hypercube (which is not yet labeled) and relabel the nodes of $CH(m - 2^{d_1})$.

To find a valid labeling for the nodes of the d_1-dimensional hypercube, we project the labels of $CH(m - 2^{d_1})$ onto the nodes of the d_1-dimensional hypercube. That is, every node of the d_1-dimensional hypercube that is adjacent to a node v in $CH(m - 2^{d_1})$ is given the label v. Since this partial labeling of the d_1-dimensional hypercube is sequential (i.e., $m - 2^{d_1}$ out of the 2^{d_1} nodes are labeled from 0 through $m - 2^{d_1} - 1$), we can easily complete it to label all the nodes in the d_1-dimensional hypercube. Thus all the nodes of the d_1-dimensional hypercube are labeled from 0 to 2^{d_1}. The labels of the nodes in $CH(m - 2^{d_1})$ are changed by adding 2^{d_1} to their labels. That is, all the nodes in $CH(m - 2^{d_1})$ are now relabeled from 2^{d_1} to $m - 1$. Therefore, all nodes of $CH(m)$ have now been labeled from 0 to $m - 1$. This completes the proof. ■

We now compute some of the graph theoretic properties of $CH(m)$. These properties include diameter, node and edge connectivity, chromatic number, node and edge clique number, and so on. We briefly define these properties in the appendix, and a reader may refer to any standard graph theory text book such as [Behzad et al. 1979] and [Bondy and Murthy 1976] for detailed definitions. Observe that when m is a power of two, $CH(m)$ is equivalent to a hypercube $Q_{\log m}$. Furthermore, if $m = 2^{k_1} + 2^{k_2} + \ldots + 2^{k_r}$ with

$k_1 > k_2 > \ldots > k_r$ and $\lfloor \log m \rfloor \geq r \geq 0$, then $CH(m)$ contains r disjoint complete hypercubes where the i-th hypercube has dimensions k_i. Let k be the smallest integer such that Q_k contains $CH(m)$ as a subgraph and let $m = b_{k-1}b_{k-2} \ldots b_0$ with $b_{k-1} = 1$. We thus have $k = \lceil \log m \rceil$.

Lemma 2. *The number of edges e in a composite hypercube CH(m) is*

$$\sum_{j=0}^{k-1} jb_j 2^{j-1} + \sum_{j=0}^{k-2} \left\lfloor b_{j+1} \sum_{i=0}^{j} b_i 2^i \right\rfloor .$$

Proof. The number of edges in $CH(m)$ is the maximum number of edges in an induced subgraph of Q_k having m nodes. The first term in the formula counts the number of edges in the disjoint hypercubes Q_js that are contained in $CH(m)$. The second term counts the number of edges that connect a hypercube Q_j with a hypercube Q_i, for all possible pairs j and i. An induction argument easily proves the formula for e and hence we omit it. ■

Lemma 3. *Let $[u, v]$ be a diametrical pair of $CH(m)$. Then there exists a partition of Q_k into Q_{k-1}^0 and Q_{k-1}^1 such that*

1. *node u is a node of Q_{k-1}^0 and node v is a node of Q_{k-1}^1, and*
2. *the distance between u and v is k.*

In other words, the diameter d of $CH(m)$ is $k = \lceil \log m \rceil$.

Proof. This lemma follows by a routine induction on k and the following two observations about hypercubes Q_k: (1) The diameter of Q_k is k, and (2) for every node x of Q_k, there exists a unique node y such that $[x, y]$ is a diametrical pair of Q_k. ■

It is well known that a hypercube Q_k contains a Hamiltonian cycle. We can easily show that the composite hypercube $CH(m)$ contains a Hamiltonian cycle when m is an even integer and it contains a Hamiltonian path when m is an odd number. Combining this with the observation that the $CH(m)$ is a bipartite graph, all the other properties of $CH(m)$ follow immediately. We tabulate these properties in Table 1. In order to compare these properties with the ones of the hypercube, we also list the properties of the hypercube $Q_{\lceil \log m \rceil}$ in the same table. As we can see from the table, composite hypercubes retain almost all properties of a hypercube. However, we must note that the drawbacks with composite hypercubes are their lack of symmetry and regularity. Experience has shown that these drawbacks can be overcome, and many algorithms designed for hypercubes can be easily modified to run on composite hypercubes [Katseff 1988; Prabhalla and Sherwani 1990; Saad and Schultz 1988].

We conclude this section by briefly considering the problem of efficiently recognizing a free composite hypercube in a complete hypercube. Given a complete hypercube Q_n of dimension n and a description of the nodes that have already been allocated to perform certain tasks, the composite hypercube recognition problem is to find m free nodes forming

Table 1. Comparison of composite hypercubes and hypercubes.

Parameter	Composite Hypercube $CH(m)$	Hypercube $Q_{\lceil \log m \rceil}$
Diameter	$\lceil \log m \rceil$	$\lceil \log m \rceil$
Min. node degree	$\Sigma_{j=0}^{k-1} b_j + i - 1$ (i is smallest integer such that $b_i = 1$)	$\lceil \log m \rceil$
Max. node degree	$\lceil \log m \rceil$	$\lceil \log m \rceil$
Node connectivity κ	$1 \leq \kappa \leq \lceil \log m \rceil$	$\lceil \log m \rceil$
Edge connectivity λ	$1 \leq \lambda \leq \lceil \log m \rceil$	$\lceil \log m \rceil$
Max. clique size	2	2
Node chromatic number	2	2
Edge chromatic number	$\lceil \log m \rceil$	$\lceil \log m \rceil$
Covering number	$\left\lceil \frac{m}{2} \right\rceil$	$2^{\lceil \log m \rceil - 1}$
Independence number	$\left\lceil \frac{m}{2} \right\rceil$	$2^{\lceil \log m \rceil - 1}$
Girth	4 (for $m > 3$)	4
Circumference	$2\left\lfloor \frac{m}{2} \right\rfloor$	$2^{\lceil \log m \rceil}$
Node clique number	$\left\lceil \frac{m}{2} \right\rceil$	$2^{\lceil \log m \rceil - 1}$
Edge clique number	e = no. of edges	no. of edges

a composite hypercube $CH(m)$ in Q_n, for any given $m < 2^n$. This problem is of critical importance in task allocation and scheduling algorithms and is similar to hypercube recognition problems [Chen and Shin 1987]. The exponential number of subcubes in a hypercube makes this problem computationally difficult. Furthermore, the dynamic nature of allocating and deallocating tasks to nodes in a hypercube algorithm leads to the fragmentation of the hypercube; that is, a total of m free nodes may be available but they may not form a composite hypercube. An allocation strategy minimizing fragmentation depends on an efficient recognition algorithm. We simply note here that the buddy tree strategy for the hypercube recognition problems [Chen and Shin 1987] can be extended to efficiently solve the composite hypercube recognition problem as well. For further details we refer the interested reader to [Boals et al. 1992].

3. Algorithms for Composite Hypercubes

Composite hypercubes are subgraphs of complete hypercubes, and it may initially appear that many algorithms for complete hypercubes should carry over directly to composite hypercubes. However, since the number of processors in a composite hypercube is arbitrary (not necessarily a power of two) and the degrees of the processors may not be equal, some of the structural symmetry of the hypercube is lost. Many known algorithms for hypercubes depend on the regularity or symmetry of complete hypercubes; in general, these algorithms cannot be used for composite hypercubes. Therefore, new algorithmic techniques need to be developed for composite hypercubes.

In this section we present algorithmic results for composite hypercubes by presenting a class of algorithms called fully normal algorithms (FNA). First we describe FNAs and their implementation on complete hypercubes. Next we develop FNAs for composite hypercubes. This is based on a mapping scheme that allocates subproblems to processors such that data flow is maintained and no processor is deadlocked. We prove that fully normal algorithms on composite hypercubes can achieve a performance that is very close to the theoretical optimum for a wide range of composite hypercube sizes. Finally, we investigate the size/performance tradeoff for composite hypercubes and empirically show that the number of extra processors needed by a composite hypercube to effectively compete with a complete hypercube decreases logarithmically, as we increase the size of the hypercube.

3.1. Fully Normal Algorithms

In this section we give a brief overview of an FNA for complete hypercubes, which will serve as a basis for the development of an FNA for composite hypercubes.

FNAs are based on *dimension collapsing*. The idea is to transfer data from one-half of the hypercube to the other half across a dimension. The given problem is partitioned into several subproblems that are assigned to the processors. In a computation cycle the processors solve their assigned subproblems and then send their results across a dimension and the processors that become free are assigned new subproblems.

In a hypercube Q_d, we say that dimension t, $0 \leq t \leq d-1$ is *collapsed* if each processor P_i with address $a_{d-1}a_{d-2} \ldots a_{t+1}1a_{t-1} \ldots a_0$ sends its results to processor P_j with address $a_{d-1}a_{d-2} \ldots a_{t+1}0a_{t-1} \ldots a_0$. Processor P_i is referred to as the t-th dimensional neighbor of processor P_j. For a hypercube Q_d, a sequence of d steps in which successive dimensions 0, 1, ..., $d-1$ are collapsed is called a *computation cycle*. For example, in Q_3, when dimension 0 is collapsed, processors 1, 3, 5, 7 send their data to processors 0, 2, 4, 6, respectively. Next we collapse dimension 1 and processors 2 and 6 send their data to processors 0 and 4, respectively. Finally, processor 4 sends its data to 0, thus completing the computation cycle. An example of this data flow is shown in Figure 2.

However, if the number of problems is greater than the number of processors in the hypercube, then we need to assign new problems to processors 1, 3, 5, 7 when they are released after the first step. A processor is considered *released* if it completes its computation, communicates its result to its dimensional neighbor, and is free to accept another problem for computation. In assigning new problems to released processors, we must make sure that

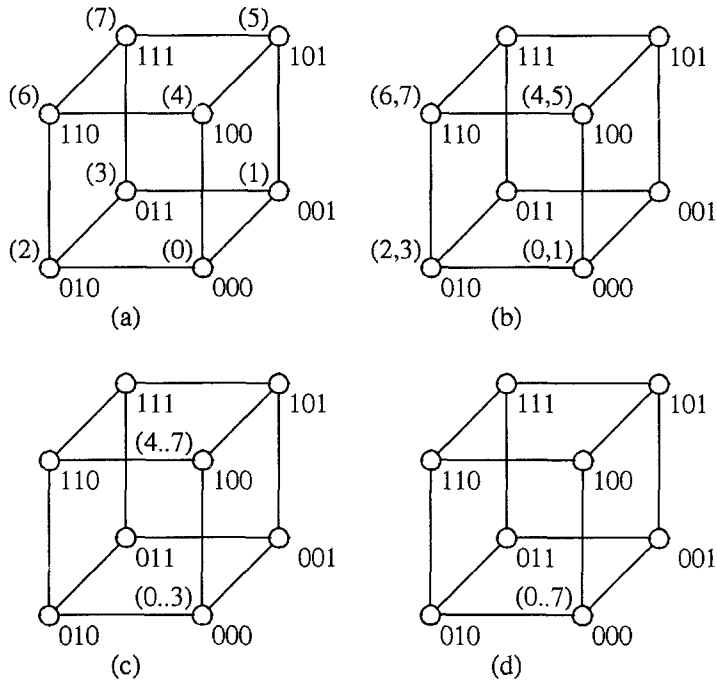


Figure 2. Data flow, when the number of problems is less than or equal to the total number of processors.

new problems also converge and no processor is deadlocked waiting for data. Reassignment schemes were developed for complete hypercubes in [Roy et al. 1989]. Here we just outline the algorithm and state the results. Informally, the algorithm works by partitioning a problem P into several subproblems and allocating the subproblems to the available processors on a dynamic basis. Any processor terminates its repeat-until loop if it does not have a subproblem to solve and it does not receive a partial solution from its dimensional neighbor. These two conditions basically reflect each processor's role as a sender or a receiver of subproblems, respectively. The important point is that the direction of the process flow must be maintained. An example of the allocation process is shown in Figure 3. It is shown in [Roy et al. 1989] that the algorithm presented above is optimal and produces correct results.

A fully normal algorithm consists of a sequence of computational cycles and halts when the final solution is in node 0. Fully normal algorithms represent a general class of algorithms used in parallel computation to solve several types of problems on hypercubes. For example, several graph problems like bipartiteness, fundamental cycles, bridges, connected components, and minimum cost spanning forests [Das et al. 1990] use fully normal algorithms. Fully normal algorithms have also been used in other areas, such as routing and compaction in VLSI design and computational geometry [Roy et al. 1989].

It is easy to see that an FNA for K subproblems running on a d-dimensional hypercube would terminate in d steps if $K \leq 2^d$. Since half of the processors are reassigned new subproblems in each step, the number of steps required for $K > 2^d$ is simple $d + \lceil K - 2^d/2^{d-1} \rceil$.

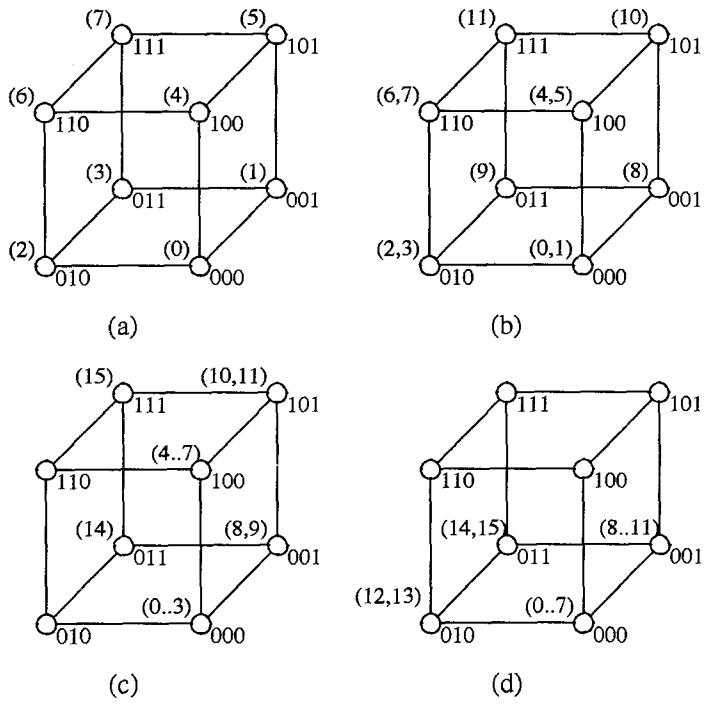


Figure 3. Data flow, when the number of problems is greater than the number of processors.

3.2. Fully Normal Algorithms for Composite Hypercubes

There are two fundamental problems that need to be resolved for the development of a fully normal algorithm for composite hypercubes.

1. *The number of available processors*: The first problem is the determination of the number of *released processors* in a given step i of the computation cycle. In the case of fully normal algorithms on a complete hypercube Q_d, 2^{d-1} processors are released in each computation step. However, in the case of $CH(m)$, $2^{d_1} < m < 2^{d_1+1}$, the number of processors released can vary from 0 to $2^{d_1} - 1$, depending on the dimension being collapsed. Let L be the list of addresses of processors in $CH(m)$ and let $SUM(i) = \Sigma_{j=0}^{m-1} b_{ij}$ where $b_{(d-1)j} b_{(d-2)j} \ldots b_{0j}$ is the address of a processor in L. The number of released processors in the i-th step of the algorithm can be computed by counting processors with the i-th bit in address set to 1, because in the i-th step, the processors that have their i-th bit = 1 send their computed subsolution to their dimensional neighbors. So these processors, after sending, become free and are thus released.

Lemma 4. *Let $RN[i]$ be the number of processors released in the i-th step of the computation cycle; then $RN[i] = SUM(i)$, $0 \leq i \leq d - 1$.*

2. *The relative ordering of the processors*: The second problem concerns the relative ordering of the released processors in a computation step. By *relative ordering*, we mean the order or sequence in which the processors are assigned new processes. The basic idea here is to reorder (or renumber) the processors so that they can easily be assigned to the next set of problems. This relative ordering is a function of the computation step and the processor number.

We solve these two problems in algorithm ALLOCATE, which allocates a problem to a released processor. Let us assume that we have an m-processor composite hypercube, $CH(m)$ and $m = 2^{d_1} + 2^{d_2} + \ldots + 2^{d_k}$ with $d_1 > d_2 > \ldots > d_k$. ALLOCATE with input $CH(m)$ initializes two arrays: RELEASED_NODES(RN[step-no]) and DIMENSIONAL_NEIGHBOR(DN[step-no][processor-no]). The largest number assigned to a processor (i.e., $m - 1$) is broadcasted to all the processors of $CH(m)$ at the beginning of the algorithm.

We need two functions, rotate and complement, to compute the array entries. Let $a = a_{n-1}a_{n-2} \ldots a_0$. We define $ROTATE(a, j)$ to be the right rotation function that rotates a to the right by j bits; that is, $ROTATE(a, j) = a_{j-1} \ldots a_1a_0a_{n-1} \ldots a_{j+1}a_j$. The *rotation* of a graph is accomplished by applying the same rotation function to the addresses of each of its processors. It can easily be seen that all adjacencies are preserved under rotation. We also define $COMP(a, j)$ as the number obtained by complementing the j-th bit of a; that is, $COMP(a, j) = a_{n-1}a_{n-2} \ldots \bar{a}_j, \ldots, a_1a_0$.

As stated earlier, the second problem that arises in allocation is to determine the relative ordering of the processors that form a composite hypercube. In the case of a complete hypercube, it can be done by rotating the current address right by i bits and complementing the $(d - 1)$-th bit. However, due to a variable number of released processors in each cycle caused by several missing processors, this procedure cannot be used in a composite hypercube. Our main observation here is that for a composite hypercube, the renumbering of processors (for each dimension) can be done *a priori*, as explained below. Suppose $CH(m)$ has been extended to a $d_1 + 1$-dimensional hypercube. Let $RN(i)$ be the list of processors released in step i. Our algorithm first reorders all the processors in $RN(i)$ as if all the processors in the $d_1 + 1$-dimensional hypercube are present. We then remove all the "nonexistent" processors from this list and refer to the resulting order among the remaining processors as the allocation order. If a processors occupies the i-th position in this list, it will be assigned the i-th subproblem from the next set of subproblems to be solved. We compute this number for each processor released in each dimension and store it in the array DN. Thus DN[i][j] denotes the sequence number of the j-th processor in the j-th processor in the $CH(m)$ released in the i-th step.

The parallel algorithm FNA_COMPOSITE is stated formally below:

Algorithm FNA_COMPOSITE:
/* This algorithm runs on each processor of the Composite Hypercube */

1. If root-processor (i.e., processor 0) then
   form subproblems P_i, $i = 0, K$
   /* Where K is the maximum number of subproblems */
2. step-no = 0.

3. P_i = ALLOCATE(step-no, i)
4. S_i = SOLVE(P_i)
5. Repeat
   5.1. if ($b_{\text{step-no}} = 0$) then
        dm = FIND-NEIGHBOR(step-no, i)
        if ($dm \neq$ Null)
          RECEIVE(dm, S_j) /* receive S_j from dimensional neighbor */
          S_i = MERGE(S_i, S_j)
        endif
   5.2. if ($b_{\text{step-no}} = 1$) then
        dm = FIND-NEIGHBOR(step-no, i)
        SEND(dm, S_i) /* send S_i to the dimensional neighbor */
        P_i = ALLOCATE(step-no, i)
        S_i = SOLVE(P_i)
   5.3. step-no = (step-no + 1) mod $\lceil \log m \rceil$
6. Until (subsolution received S_j = NULL) OR
   (new subproblem P_i = NULL)

ALLOCATE(step-no,processor-no)
/* P_α is the largest numbered subproblem assigned in step (step-no − 1) */

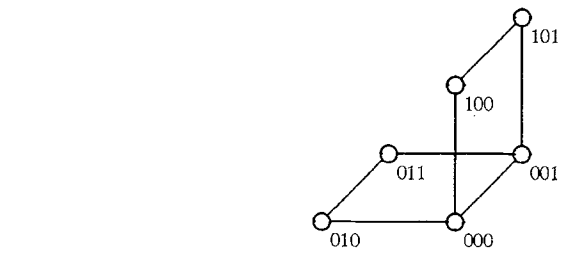
1. If step-no = 0 then
   $\alpha = m + 1$
   return $P_{(processor\text{-}no)}$
2. else
   offset = DN[step-no][processor-no]
   $P_{\alpha'} = P_{(\alpha+offset)}$
   $\alpha = \alpha$ + RN[step-n]
   return $P_{\alpha'}$

FIND-NEIGHBOR(step-no,processor-no)

1. dm = COMP(processor-no, step-no)
2. If ($dm \leq m$) then (return dm) else (return NULL);

The example in Figure 4 shows how the dimension collapsing and processor reallocation works in a composite hypercube $CH(m)$ along with the details of the two global arrays $RN[\ ]$ and $DN[\ ][\ ]$ that control the allocation order.

3.2.1. Analysis of Algorithm FNA_COMPOSITE. Assume that m is the number of processors in the composite hypercube, $d = d_1 + 1$ is the dimension of the bounding complete hypercube, and $N = 2^d$ is the number of processors in the bounding complete hypercube. Let K be the number of subproblems. Two cases arise in this scheme: First, $K \leq m$. In this case the allocation is static in the sense that the subproblems are allocated only once and the problem is solved in one computation cycle. Second, $K > m$. Here all the



Dim: 0 1 2

3	2	2

Relaxed node array

Node numbers

Dimension

	0	1	2	3	4	5
0		0		1		2
1			0	1		
2					0	1

Dimensional neighbor aray

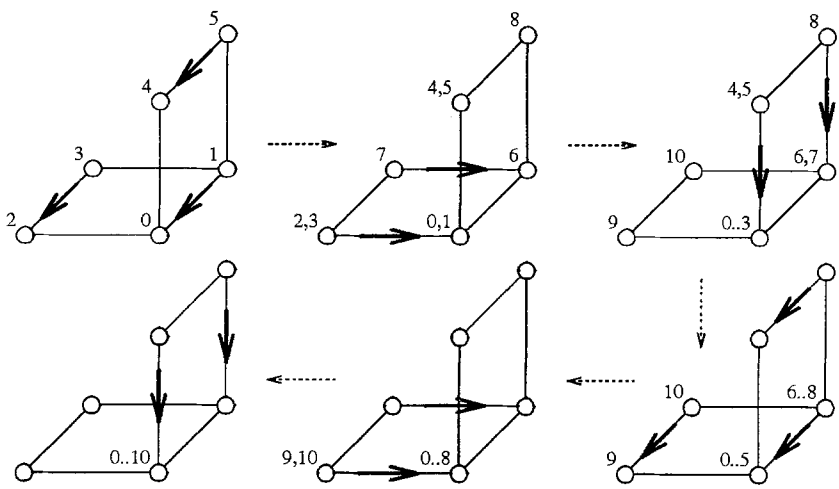


Figure 4. An example of the algorithm FNA_COMPOSITE.

subproblems cannot be allocated to processors and the computation proceeds in several cycles, and in each computation cycle we must allocate the remaining subproblems to the released processors in a pipelined fashion. The allocation scheme must address the problem of reallocation of released processors. Let us examine these two cases in detail.

Allocation scheme for $K \leq m$: In this case no reallocation is needed since the number of processors is sufficient to solve all the subproblems after the initial allocation. Each processor i is initially assigned subproblem P_i, which it solves using procedure SOLVE to obtain the subsolution S_i. In step 0, each processor with its 0-th bit equal to 1 sends

its subsolution to its dimensional neighbor that contains the adjacent subsolution. The neighbor is found by complementing the 0-th bit of i. For example, processor 001 sends to processor 000, and processor 011 sends to processor 010. In the same step (i.e., step 0), each receiving processor i merges its subsolution S_i with its received subsolution S_{i+1} to obtain $S_{(i,i+1)}$. In the next step this result is sent to the processor that contains $S_{(i-2,i-1)}$.

In general, in step $j < d_1 + 1$, each processor $s = b_d b_{d-1} \ldots b_j \ldots b_1 b_0$ with $b_j = 1$ sends its subsolution $S_{(p,q)}$ to processor $r = COMP(s, j)$, while each processor $r = b_d b_{d-1} \ldots b_j \ldots b_1 b_0$ with $b_j = 0$ receives a subsolution $S_{(p,q)}$ from processor $s = COMP(r, j)$ and merges it with its subsolution $S_{(r,p-1)}$ to obtain $S_{(r,q)}$. Similarly, in step $\lceil \log m \rceil - 1$, which is the last step, processor 0 receives subsolution $S_{(N/2,N-1)}$ from the processor $N/2$ and merges the received solution with its own subsolution $S_{(0,N/2-1)}$ to obtain $S_{(0,N-1)}$. Because $S_{(0,N-1)}$ defines the solution to the entire problem, the algorithm terminates.

Lemma 5. *Let problem P be divided into K subproblems P_i, $1 \leq i \leq K$. Then, for $K \leq m$ the algorithm FNA_COMPOSITE solves P in $\lceil \log m \rceil$ steps.*

Proof: The proof in this case follows from Lemma 4 for complete hypercubes since there is no reallocation of released processors. However, since some of the processors are missing, we need to show that the subsolutions arrive in the correct sequence at any processor. This follows from the fact that in any step all the sending processors have higher addresses than their corresponding receiving processors and all the processors in a composite hypercube are sequentially numbered from 0 to $m - 1$. Since there are sending processors in all the dimensions 0 through $\lceil \log m \rceil$, problem P is solved in $\lceil \log m \rceil$ steps. ■

Allocation scheme for $K > m$: We now consider the case when the number of processors is less than the number of subproblems, that is, $m < K$. In this case, only m out of the K subproblems can be assigned to processors in the first step, and we need to consider reallocation of the released processors. First, we show in the case of a $CH(m)$ that the released processors always form a composite hypercube that has a "local zero" processor.

Lemma 6. *If R is the set of processors released in the i-th step of the algorithm, then the processors in R form a composite hypercube $CH(RN(i))$. Furthermore, there exists a processor $r \in R$, such that r is adjacent to processor zero of $CH(m)$. We call this node r the "released root."*

Proof: According to the definition of composite hypercube $CH(m)$, $m = \Sigma_{i=1}^{k} 2^{d_i}$, we can view it as a complete hypercube Q_{d_1} of dimension d_1 and a composite hypercube $CH(m - 2^{d_1})$. If the collapsing dimension is one which connects these two components, then $CH(m - 2^{d_1})$ is released. If the collapsing dimension is other than the connecting dimension, then the complete hypercube Q_{d_1} releases $2^{d_i - 1}$ processors, which form a complete hypercube of dimension $d_1 - 1$. All processors released by $CH(m - 2^{d_1})$ have dimensional neighbors in this $(d_1 - 1)$-dimensional hypercube and hence the processors in R form a composite hypercube.

In an incomplete hypercube some addresses with $b_i = 1$ may not be the addresses of existing processors. However, if there exists *any* processor in the current collapsing dimen-

sion, then the processor with $b_i = 1$ and all other bits zero must be present since it has the least address among all processors in this dimension. This processor is adjacent to processor zero. We refer to this processor as the "released root" and denote it by r_{si}. ■

Next we show that in the $CH(m')$ formed by m' released processors, the subsolutions from the reallocated processors converge onto their "local zero" processor in $\lceil \log m' \rceil$ steps.

Lemma 7. *Assume that in a reassignment step i in ALLOCATE, m' subproblems are assigned to a released composite subcube of dimension $d' = \lceil \log m' \rceil$; then processor r_{si}, the root processor of this composite subcube, will contain the solution to this set of subproblems in d' steps.*

Proof: The reassignment scheme in ALLOCATE results in a subcube of m' "released" processors that can be renumbered starting from 0 using $RN[i]$, and $DN[i][0 \ldots m' - 1]$ such that r_{si} is labeled as 0. According to the above lemmas this assignment will result in the solution $S_{(l,l+m')}$ being computed at processor r_{si} in $\lceil \log m' \rceil$ steps. ■

Theorem 1. *Algorithm FNA_COMPOSITE for a composite hypercube terminates successfully with the result in processor zero.*

Proof: After the initial set of assignments of problems $P_0, P_1, \ldots, P_{m-1}$ in step 0, new subproblems are assigned at each step and the solution to each set of these subproblems reaches the root processor of their subcube r_{si} as per Lemma 7. In the next step these solutions are merged with the existing solution at processor zero of $CH(m)$. ■

Finally note that Algorithm FNA_COMPOSITE is well defined and no processor is blocked waiting for data from an absent processor. Two important features of the algorithm FNA_COMPOSITE guarantee this condition. First, the function FIND_NEIGHBOR returns NULL in case the computed neighbor has an address greater than the $m - 1$. Otherwise, it returns the processor's address. Second, the receiving processor executes only if a neighbor in the executing dimension is present to complete the pairwise communication. These two steps make sure that a receiving processor is never blocked expecting to receive a subsolution from an absent processor. Sending processors are never blocked since they always send their data to the processors with lower numbers, and it follows from the definition of composite hypercubes that all lower numbered processors are present.

3.3. Size/Performance Tradeoffs for Composite Hypercubes

With respect to FNAs, complete hypercubes have a simple linear performance behavior. That is, Q_{d+1} can process twice as many subproblems as Q_d in the same number of steps. This is due to the uniform and symmetric structure of complete hypercubes. In the case of composite hypercubes the speedup calculation is complicated by the fact that the number of processors released in each step may be different. In this section we compute the speedup that can be obtained by a $CH(m)$. It will be shown that over most of the range $2^d, 2^d + 1, \ldots, 2^{d+1}$ composite hypercubes exhibit performance behavior close to Q_d. For example,

we show that a $CH(1536)$ can process 1.45 times as many subproblems as a complete hypercube with 1024 processors. The ideal number would be 1.5. This means that $CH(1536)$ is only 6% less efficient than an ideal 1536-processor architecture. Therefore, the critical parameter is the number of subproblems that can be processed by a composite hypercube in a fixed number of steps. We refer to this parameter as *PN*.

With respect to fully normal algorithms, the *PN* may be calculated in terms of the number of released processors in a computation cycle, since the number of subproblems solved is equal to the number of processors released. Since we compare complete hypercubes with composite hypercubes, we define the relative processors number (RPN) achieved by an FNA on $CH(m)$ to be the ratio of the number of processors released by $CH(m)$ in a $d_1 + 1$ step computation cycle and the number of processors released by Q_{d_1} in $d_1 + 1$ steps, where $m = 2^{d_1} + 2^{d_2} + \ldots + 2^{d_k}$. The $d_1 + 1$ step computation cycle must be considered for Q_{d_1} since the number of steps in the computation cycle of $CH(m)$ is one more than that of Q_{d_1}.

In order to obtain an expression for the RPN, we need to compute the number of processors released by a complete hypercube in a computation cycle. Let $R(m)$ be the number of processors released by a $CH(m)$ in a computation cycle.

Lemma 8. *For a d-dimensional complete hypercube, the number of released processors in a computation cycle is given by* $R(m) = d * (m/2)$ *where* $m = 2^d$.

Proof: The proof follows from the fact that each dimension collapse releases $m/2$ processors and there are d dimensions. ■

Thus the RPN for $CH(m)$ (namely $RPN(m)$) may be defined as

$$RPN(m) = \frac{R(m)}{R(2^{d_1}) + \dfrac{2^{d_1}}{2}}$$

$$RPN(m) = \frac{R(m)}{(d2^{d_1-1}) + \dfrac{2^{d_1}}{2}}$$

The second term in the denominator expresses the number of processors released in the $(d_1 + 1)$-st step of the computation cycle of Q_{d_1}. Before we compute *RPN* for the general case, it is illustrative to compute $RPN(m)$ for a $CH(m)$ when $m = 2^d + 2^{d-1}$, that is, for a composite hypercube consisting of two complete hypercubes of successive sizes. Let us first compute the number of processors released by a $CH(m)$ in this case.

Lemma 9. *The number of processors released in a computation cycle of* $CH(m)$ *is given by* $R(m) = d * 2^{d-1} + (d - 1) * 2^{d-2} + 2^{d-1}$ *and* $RPN(m) = \frac{3d+1}{2d+2}$ *if* $m = 2^d + 2^{d-1}$.

Proof: We may consider $CH(m)$ as composed of two complete hypercubes of dimensions d and $d - 1$. During the collapse of the first $d - 1$ dimension, both hypercubes release half of their processors independently. In the d-th step, Q_d releases half of its processors; however, Q_{d-1} does not have any processors in this dimension and hence releases no processors. Thus Q_d releases 2^{d-1} processors for d steps, while Q_{d-1} releases 2^{d-2} processors for $d - 1$ steps. The third term in the expression represents the release of all the processors of Q_{d-1}, while the $d + 1$ dimension is collapsed.

The $RPN(m)$ in this case is computed as follows:

$$RPN(m) = \frac{R(m)}{R(2^d) + \frac{2^d}{2}} = \frac{3d + 1}{2d + 2}$$

This completes the proof. ■

Note that this type of hypercube is quite efficient and the efficiency is better for larger sizes. For example, with $d = 5$, $RPN(48) = 1.33$, whereas with $d = 10$, $RPN(1536) = 1.45$. This shows that $RPN(48)$ ($RPN(1536)$) is only 11% (resp. 6%) slower than its idealized counterpart with the same number of processors.

In order to generalize the result in the above lemma to an arbitrary m, we provide the following theorem.

Theorem 2. *The number of processors released by a composite hypercube $CH(m)$ in a computation cycle is given by*

$$R(m) = \sum_{i=1}^{k} d_i 2^{d_i - 1} + \sum_{i=1}^{k-1} i 2^{d_{i+1}}.$$

Proof: As per the definition of a composite hypercube, one may consider a $CH(m)$ as consisting of two components: a d_1-dimensional complete hypercube and a $CH(m - 2^{d_1})$. Let r be the processors in the *noncomplete* part of $CH(m)$; that is, $r = m - 2^{d_1}$. $CH(m)$ has a $d_1 + 1$ step computation cycle. In the first d_1 steps both components compute independently. It is clear that a d_1-dimensional complete hypercube will release $2^{d_1 - 1}$ processors in each cycle, thereby releasing a total of $d_1 2^{d_1 - 1}$ processors in d_1 steps. On the other hand, the number of processors released by a $CH(r)$ is given by $R(r)$ and may be computed recursively. In step $d_1 + 1$, the last dimension (i.e., $d_1 + 1$) is collapsed and all the processors in $CH(r)$ are released. Therefore, the number of processors released by $CH(m)$ can be expressed by the following recurrence relation:

$$R(m) = d_1 2^{d_1 - 1} + R(r) + r.$$

Note that $R(r) = d_2 2^{d_2 - 1} + R(r - 2^{d_2}) + r - 2^{d_2}$; substituting we get

$$R(m) = d_1 2^{d_1 - 1} + (d_2 2^{d_2 - 1} + R(r - 2^{d_2}) + r - 2^{d_2}) + 2^{d_2} + 2^{d_3} + \ldots + 2^{d_k}.$$

Solving this recurrence relation we obtain the desired result. ■

The *RPN*(m) for *CH*(m) can now be easily computed.

Corollary 1. *The RPN achieved by a composite hypercube CH*(m) *is given by*

$$RPN(m) = \frac{\Sigma_{i=1}^{k} d_i 2^{d_i - 1} + \Sigma_{i=1}^{k-1} i 2^{d_{i+1}}}{(d_1 + 1)2^{d_1 - 1}}.$$

Figure 5 shows the *RPN* of a FNA on composite hypercubes *CH*(m), $m = 32, 33, \ldots, 128$.

It may be seen from Figure 5 that not all composite hypercubes are *useful*. That is, some perform worse than their own (largest) component complete hypercubes. For example, *CH*(33) has an *RPN* of 0.844, as compared to a 32-processor complete hypercube. This implies that the effectiveness of the architecture is degraded by the addition of a single processor. This is due to the fact that the extra dimension does not have *enough* processors to *justify* the cost of computation in that dimension.

In this context it is natural to consider the question of the minimum number of additional processors needed so that the resulting composite hypercube performs better than the original complete hypercube. Consider the *RPN*s for the composite hypercubes *CH*(33) through *CH*(40) listed in Table 2. It can be seen that *CH*(39) is the smallest composite hypercube that performs equal to or better than a 32-processor complete hypercube. We

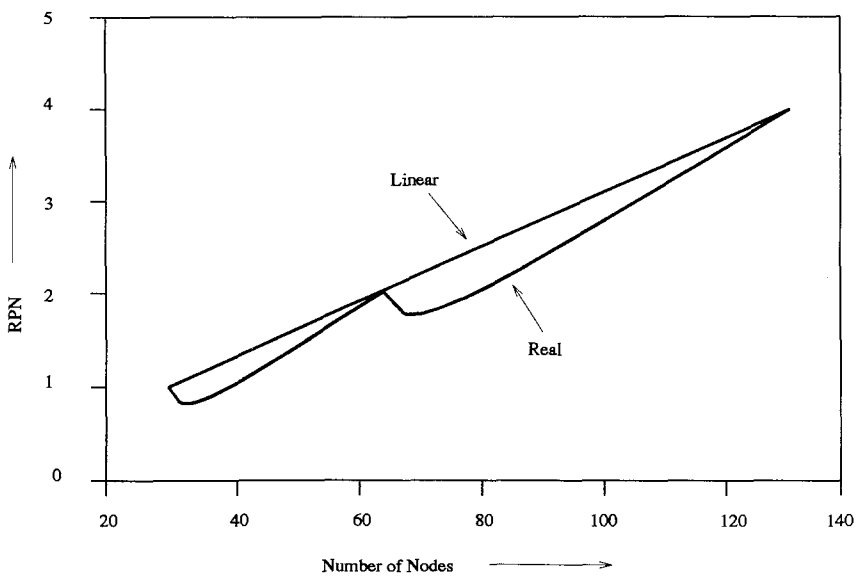


Figure 5. RPN of an FNA on a composite hypercube.

*Table 2. RPN*s for the composite hypercubes *CH*(33) through *CH*(40).

Processors m	32	33	34	35	36	37	38	39	40
RPN(m)	1.000	0.844	0.865	0.885	0.917	0.938	0.969	1.000	1.042

Table 3. Critical composite hypercubes and their respective critical number of processors.

Processors 2^d	4	8	16	32	64	128	512	1024
Crit. comp. hypercube $CCH(m')$	6	11	20	39	76	150	576	1143
Crit. no. of procs. $CN(2^d)$	2	3	4	7	12	22	64	119
% procs. needed	50%	37%	25%	22%	19%	17%	13%	12%

call such a composite hypercube a *critical composite hypercube* $CCH(m')$ and refer to $CN(2^d) = m' - 2^d$ as the critical number of processors (see Table 3.) In Table 3 we also show the number of processors (in terms of percentage) needed to reach criticality. It can be observed that the number of processors needed decreases for larger hypercubes, showing that criticality is easier to achieve for larger hypercubes.

The data in Table 3 leads us to make the empirical observation that the minimum number of processors needed to reach the critical composite hypercube, $t = m - 2^{d_1}$, decreases logarithmically with d_1, which is exhibited in Figure 6. We note that a number of experiments covering larger values of the number of processors were run; however, the table lists only a few data values that we consider to be practical.

It is worthwhile to note that the *RPN* of composite hypercubes is very close to the ideal architecture, after the addition of the critical number of processors.

4. Embeddings into Composite Hypercubes

One of the main reasons for the popularity of hypercube architecture is its ability to simulate other architectures very efficiently. If composite hypercubes are to be competitive as an architecture, we must demonstrate similar simulation capabilities. Graph embeddings have been used very successfully to show simulation capabilities of a guest architecture by another host architecture [Aleliunas and Rosenberg 1982; Bhatt and Ipsen 1985; Chan 1988;

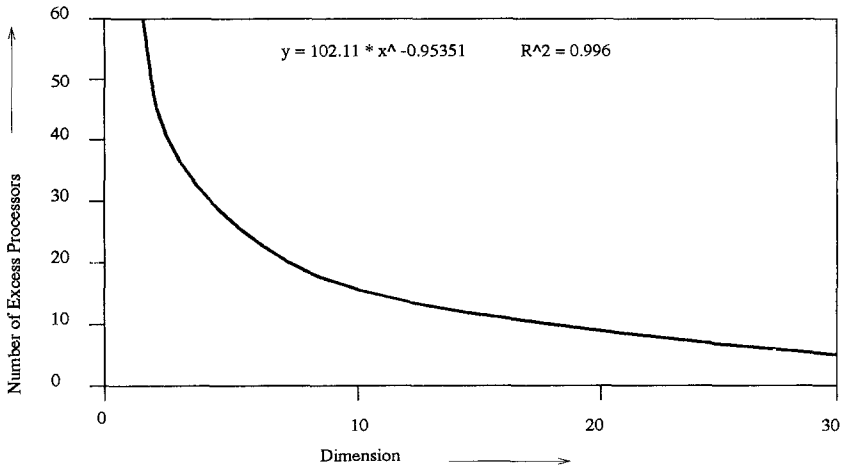


Figure 6. The number of processors needed to reach criticality.

Fishburn and Finkel 1982; Gupta 1989; Hong et al. 1983; Kosaraju and Atallah 1988]. In graph embedding techniques, host and guest architectures are viewed as graphs H and G, respectively, and then the graph G is embedded into the graph H. In the embedding of a graph G into H, we injectively map the set of nodes of G into the set of nodes of H and the edges of G to paths in H that connect the image of the nodes of G. In order to obtain efficient simulations of G by H, various cost measures of an embedding must be optimized. Three such measures are the *dilation*, the *congestion* and the *expansion* of an embedding. The dilation of an edge of G is the length of the path onto which an edge of G is mapped. The dilation of the embedding is the maximum edge dilation of G. The congestion of an edge in H is the number of paths passing through it, and the congestion of the embedding is the maximum congestion of any edge in H. The expansion of the embedding is the ratio of the number of nodes in H to the number of nodes in G. Ideally, we would like to find embeddings with minimal dilation, congestion and expansion so that efficient simulations of G by H can be obtained. If an embedding achieves the minimum expansion, then we say that the embedding uses an optimal size H. Throughout this paper we consider embeddings with a dilation of 1 or 2, and the resulting embeddings achieve a congestion of 1 or 2. Hence without loss of generality we suppress the discussion of congestion and only give our results with respect to dilation and expansion.

Embeddings of binary trees, two-dimensional meshes, butterflies and cube-connected cycles into complete hypercubes have been studied extensively [Bhatt and Ipsen 1985; Bhatt et al. 1986; Chan 1988; Greenberg et al. 1990; Ho and Johnsson 1990]. The main reason for studying embeddings of these architectures is their suitability for the development of specific types of parallel algorithms. For example, mesh architectures are well suited for scientific and computer vision applications, whereas tree machines are well suited for divide-and-conquer type algorithms. Complete binary trees, two-dimensional meshes and cube-connected cycles are shown to be embeddable into complete hypercubes with a dilation of 2 and an expansion of less than 2 [Bhatt and Ipsen 1985; Chan 1988]. In [Greenberg et al. 1990] it is shown that butterflies are subgraphs of their optimal-sized complete hypercubes (the smallest hypercube having at least as many nodes as the butterfly); that is, a butterfly can be embedded with dilation 1. Very few results are known about the embeddings of these architectures into incomplete or composite hypercubes. To the best of our knowledge, only Tzeng et al. [1990] have investigated the embeddings of complete binary trees, incomplete binary trees and meshes into their definition of incomplete hypercubes.

Due to the fact that composite hypercubes are subgraphs of complete hypercubes, it may initially appear that the embedding strategies for complete hypercubes should carry over directly to composite hypercubes. However, since the number of nodes in a composite hypercube is arbitrary (not necessarily a power of two) and the degrees of the nodes need not be equal, some of the structural symmetry is lost. Since many known embedding strategies depend on the regularity or symmetry of complete hypercubes, in general these strategies cannot be used for composite hypercubes. Therefore, new embedding strategies are needed. One of the goals of this paper is to investigate new strategies for embeddings of binary trees, two-dimensional meshes, butterflies and cube-connected cycles into composite hypercubes.

It can be easily shown that an n-node complete binary tree can be embedded into an n-node composite hypercube with a dilation of 2 using the ideas of [Bhatt and Ipsen 1985] since $n = 2^h - 1$. Several divide-and-conquer type algorithms exhibit incomplete binary

tree structure; thus it is interesting and natural to investigate the classes of incomplete binary trees that can be efficiently embedded into composite hypercubes. Section 4.1 shows the embeddings of various types of n-node incomplete binary trees into n-node or $(n + 1)$-node composite hypercubes with a dilation of at most 2. We also present lower bound proofs showing the optimality of the dilation. We also characterize the class of incomplete binary trees that are subgraphs of composite hypercubes. Efficient embeddings of X and leap trees are also investigated.

In Section 4.2 we present a dilation-1 embedding of a two-dimensional n-node mesh into its optimal n-node composite hypercube, where one dimension is a power of two. When neither dimension is a power of two, it is shown that a dilation-1 embedding is not possible, thereby characterizing the class of two-dimensional meshes that can be embedded into composite hypercubes with dilation 1. All two-dimensional meshes are shown to be embeddable with dilation 1 if expansion greater than 1 but less than 2 is allowed. We also consider two types of incomplete meshes and their embeddings into their optimal composite hypercubes. Section 4.3 considers embeddings of n-node butterflies and cube-connected cycles into n-node composite hypercubes, and the dilation of these embeddings is proved to be optimal.

4.1. Embedding Trees

In this section we consider embeddings of complete binary trees, incomplete binary trees, X-trees and leap-trees into composite hypercubes. Researchers have previously considered embeddings of these treelike networks into complete hypercubes. From now onward, for reasons of clarity, we will refer to the nodes of a composite or a complete hypercube as *PE*s (processing elements).

An n-node complete binary tree T_h, of height h, can be embedded into a complete hypercube Q_{h+1} of $(n + 1)$-PEs with dilation 2 for $n = 2^{h+1} - 1$. There are many embeddings that achieve these bounds on embeddings. One such simple embedding is *inorder* embedding [Gupta 1989], which labels the nodes of T_h using an inorder traversal (the leftmost leaf of T_h is labeled as 0) and then assigns the node with label i to PE i of Q_{h+1}, $0 \leq i \leq n - 1$. Observe that $(n - 1)/2$ edges of T_h are dilated by 2 in the inorder embedding. Bhatt and Ipsen [1985] showed that an $(n + 1)$-node two-rooted complete binary tree TR_h is a subgraph of Q_{h+1} and thus can be embedded with dilation 1 (TR_h is obtained from a complete binary tree T_h by replacing one of the edges incident on the root r with a path of length 2, and the additional node on the path is called the second root, say r'). Throughout this section we refer to this embedding as *BI-embedding*. This result gives an embedding of T_h into Q_{h+1} with only one edge having dilation 2 and rest of the $n - 2$ edges having dilation 1.

A natural question that arises is, How well do composite hypercubes embed binary trees? Let us first consider a complete binary tree, that is, a binary tree in which all the leaves are at the same level and all possible leaves exist. Since a composite hypercube $CH(n)$ is a subgraph of a complete hypercube Q_{h+1} and since a complete binary tree T_h cannot be embedded into Q_{h+1} with a dilation of 1, it is easy to see that T_h cannot be embedded into an n-PE composite hypercube $CH(n)$ with a dilation of 1. Furthermore, by first using the BI-embedding and then deleting the PE of Q_{h+1} that does not get a node of T_h assigned to it, the complete binary tree T_h can be embedded into $CH(n)$ with dilation 2.

Observation 1. *An n-node complete binary tree T_h can be embedded into an n-PE composite hypercube CH(n) with an optimal dilation of 2.*

The above observation implies that the problem of embedding a complete binary tree into a composite hypercube is not very interesting, at least theoretically. However, composite hypercubes are defined for any arbitrary number of PEs; hence one needs to explore incomplete binary tree structures that arise in many applications along with their embeddings into composite hypercubes. We define two types of incomplete binary trees that are denoted as Type 1 and Type 2. These types of trees arise, in general, in the applications that use the divide-and-conquer paradigm for algorithm development, and in particular, in branch-and-bound and heuristic search applications, parallel priority queue (heaps) implementations and parallel graph algorithms, such as the shortest path and spanning forest problems. Our attempt here is to consider only a few types of incomplete tree structures in the hope that they will guide us in solving the general problems when arbitrary types of incomplete tree structures arise.

Type 1 incomplete binary trees consists of a sequence of classes of binary trees, $IT1_h$ for every $h \geq 0$ where h denotes the height of the trees in a class. The class $IT1_0$ consists of a single node and $IT1_1$ consists of the two binary trees of height 1. The class $IT1_h$, for $h \geq 2$, is defined recursively as follows: The first tree of this class is a two-rooted complete binary tree TR_{h-1} and all the other trees are obtained by appending a tree from class $IT1_m$ with $m \leq h - 2$ to the root r' of TR_{h-1}. Figure 7 shows the first four classes of Type 1 incomplete binary trees. Observe that the number of trees $|IT1_h|$ in the h-th class is $|IT1_{h-1}| + |IT1_{h-2}|$, for $h > 2$. For example, the number of trees in the first seven classes are 1, 2, 2, 4, 6, 10, and 16. (Type 1 incomplete binary trees are interesting from two aspects: First, they characterize a whole class of binary trees that are subgraphs of their optimal-sized composite hypercubes, and second they contain special cases of Type 2 incomplete binary trees (essentially heaps on full binary trees), as discussed later.)

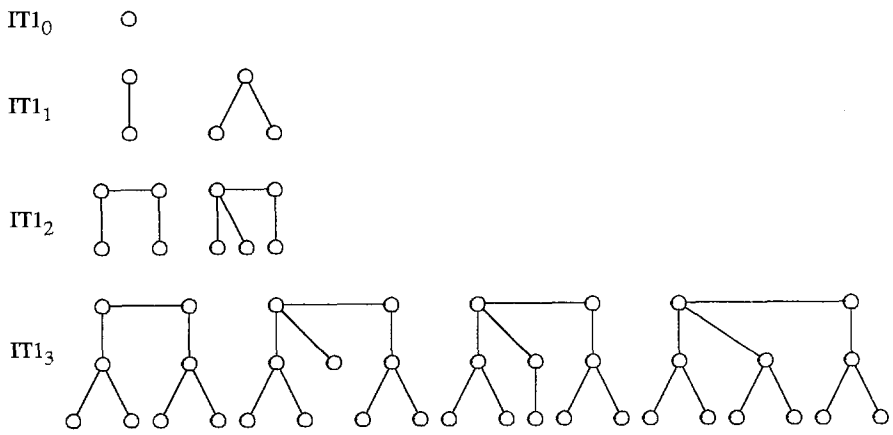


Figure 7. Four classes of Type 1 incomplete binary trees.

We next show that Type 1 incomplete binary trees are subgraphs of their optimal-sized composite hypercubes.

Theorem 3. *Every Type* 1 *incomplete binary tree can be embedded into a composite hypercube with a dilation of* 1 *and an expansion of* 1.

Proof: For the sake of brevity we give only the basic ideas behind the embedding. The reader is referred to [Boals et al. 1992] for details. We extend the ideas of Bhatt and Ipsen [1985] and use induction to give an embedding. It is clear that all the trees in $IT1_0$ and $IT1_1$ can be embedded into the optimal composite hypercubes with dilation 1. For every class $IT1_h$, for $h > 1$, the first tree is a two-rooted tree TR_{h-1}. By using BI-embedding this tree can be embedded into its optimal composite hypercube $CH(2^h)$, which is a complete hypercube Q_h, with dilation 1. In order to embed all the other trees in $IT1_h$, observe that an n-node tree T in $IT1_h$ is composed of a two-rooted complete binary tree TR_{h-1} and a tree, say T', from class $IT1_m$, $m \leq h - 2$, such that the root of T' is connected to one of the roots r of TR_{h-1}. By induction we can embed T' into its optimal-sized composite hypercube $CH(n - 2^h)$ with dilation and expansion 1. By using BI-embedding, TR_{h-1} can be embedded into its optimal-sized composite hypercube $CH(2^h)$ with dilation and expansion 1. Combining the embeddings of T' and TR_{h-1} and appropriately translating one of the embeddings establishes the theorem. ■

We now consider full trees that commonly arise in applications using heaps, and we refer to them as Type 2 incomplete binary trees. An n-node Type 2 incomplete binary tree $IT2(n)$ is a complete binary tree of height h with the rightmost $2^{h+1} - n - 1$ leaves missing, for the smallest h such that $2^{h+1} \geq n$. A common way to label a complete binary tree is to label the root as 0 and then for any node v that is labeled i, the left child of v is labeled $2i + 1$ and the right child is labeled $2i + 2$. It is thus easy to see that in a similar manner the nodes of $IT2(n)$ can be labeled from 0 to $n - 1$. Type 2 trees were also considered in [Tzeng et al. 1990] and the authors showed the existence of dilation-1 embeddings for several special cases. We note that the expansion of their embeddings is greater than 1 whereas our focus is to find embeddings into composite hypercubes with an expansion very close to 1. We next show that $IT2(n)$ can be embedded into $CH(n + 1)$ with a dilation of 2. This embedding uses only one extra PE from its optimal-sized composite hypercube. As will be shown by Lemma 10, for $n \neq 2/3(2^{h+1} - 1) + a$ when h is odd, or $n \neq 4/3(2^h - 1) + a$ when h is even, with $a = -1, 0, 1$, this embedding is optimal with respect to dilation. For the cases when this embedding is not optimal, it is easy to see that Type 2 incomplete binary trees are also Type 1 incomplete binary trees, which we have shown to be embeddable into $CH(n)$ with dilation and expansion 1.

Theorem 4. *An n-node Type* 2 *incomplete binary tree* $IT2(n)$ *is embeddable in a* $(n + 1)$*-PE composite hypercube* $CH(n + 1)$ *with an optimal dilation of at most* 2.

Proof: Observe that the tree $IT2(n)$ may be viewed as a complete binary tree T_{h-1} of height $h - 1$ and a set of $n - 2^h + 1$ nodes. The root of T_{h-1} is the same as the root of $IT2(n)$ and $n - 2^h + 1$ nodes are the leaves of $IT2(n)$ such that they are the children of the $\lceil (n - 2^h + 1)/2 \rceil$ leftmost leaves of T_{h-1}. We first embed T_{h-1} into a complete hypercube

Q_h using the inorder embedding and then embed the $n - 2^h + 1$ children of the leaves of T_{h-1} into a $CH(n - 2^h + 1)$ so that Q_h and $CH(n - 2^h + 1)$ together form a composite hypercube $CH(n + 1)$. We omit further details of this embedding strategy, for the sake of brevity, and interested readers are referred to [Boals et al. 1992] for details. ■

Lemma 10. *An n-node Type 2 incomplete binary tree $IT2(n)$ of height h cannot be embedded into an n-PE composite hypercube $CH(n)$ with a dilation of 1 for an arbitrary integer n. The only values of n where $IT2(n)$ can be embedded into $CH(n)$ with a dilation of 1 are $n = 2/3(2^{h+1} - 1) + a$ when h is odd, and $n = 4/3(2^h - 1) + a$ when h is even, with $a = -1, 0, 1$, for the smallest h such that $2^{h+1} > n > 2^h$.*

Proof: The proof relies on the following fact about bipartite graphs: The partite sets of a connected bipartite graph are unique. We know that an n-PE composite hypercube $CH(n)$ is a connected bipartite graph. As a consequence, the partite sets of $CH(n)$ are unique. Furthermore, the number of PEs in one partite set differs by at most one from the number of PEs in the other set (they differ by one when n is an odd integer). Any n-node Type 2 incomplete binary tree $IT2(n)$ is a connected bipartite graph. If the tree $IT2(n)$ can be embedded into $CH(n)$ with a dilation of 1 (that is, $IT2(n)$ is a subgraph of $CH(n)$), then the number of nodes in the partite sets of $IT2(n)$ must also differ by at most one. It is easy to see that for $n \neq 2/3(2^{h+1} - 1) + a$ when h is odd, and $n \neq 4/3(2^h - 1) + a$ when h is even, with $a = -1, 0, 1$, the partite set of $IT2(n)$ differ by at least 2. Thus for every h there are at most three n-node trees $IT2(n)$s that could be subgraphs of $CH(n)$. We next show that each one of these trees is embeddable in $CH(n)$ with dilation 1.

The embedding will follow from Theorem 3 and the fact that each of the three possible trees are also Type 1 incomplete binary trees. We establish this by induction on h, the height of $IT2(n)$. The reader can easily verify that whenever $h \leq 2$, the lemma is satisfied. Let $h \geq 3$ be the height of an n-node Type 2 incomplete binary tree $IT2(n)$ with $n = 2/3(2^{h+1} - 1) + a$ when h is odd, and $n = 4/3(2^h - 1) + a$ when h is even, with $a = -1, 0, 1$. The subtree T rooted at the right child of the left child of the root of $IT2(n)$ is a Type 2 incomplete binary tree of height $h - 2$ and the number of nodes in T is $m = 2/3(2^h - 1) + \eta$ when h is odd, and $m = 4/3(2^{h-2} - 1) + \eta$ when h is even, with $\eta = -1, 0, 1$. Using the induction hypothesis, we know that T is a Type 1 incomplete binary tree. Since $IT2(n) - T$ is a two-rooted complete binary tree of height $h - 1$ from the definition of Type 1 incomplete binary trees, it follows that the tree $IT2(n)$ is a Type 1 incomplete binary tree. Using Theorem 3, $IT2(n)$ can be embedded into $CH(n)$ with a dilation of 1 and the lemma follows. ■

In [Boals et al. 1992] another type of incomplete binary tree structure is considered, namely an n-node Type 3 incomplete binary tree $IT3_k^h(n)$ of height h. Tree $IT3_k^h(n)$ is defined for $n = 2^h + 2^k - 1$ and it consists of complete binary tree T_k of height k and 2^{k-1} complete binary trees, each of height $h - k$, with their roots as the alternate leaves of T_k. Let $\mathcal{A}$ be the class of divide-and-conquer type algorithms in which every process at a node up to level $k - 1$ in a complete binary tree T_h of height h spawns two processes and then only one of the two children of every node at level $k - 1$ spawns two processes up to level h in T_h. If an algorithm in class $\mathcal{A}$ is abstracted as a binary tree and this

algorithm is simulated by composite hypercubes, then Type 3 incomplete binary trees are of interest. Type 3 trees would also arise in situations when say m, $m = 2^{k-1} + 1$, tasks are to be run concurrently on a composite hypercube and one of the tasks corresponding to the tree T_k is the master task that interacts with the other $m - 1$ independent tasks. We have shown that except when $n = 5$ (with $h = 2$ and $k = 1$) or $n = 11$ (with $h = 3$ and $k = 2$), a tree $IT3_k^h(n)$ cannot be embedded into an n-PE composite hypercube $CH(n)$ with a dilation of 1. We also describe an embedding that embeds $IT3_k^h(n)$ for $n \neq$, 5, 11 into $CH(n)$ with a dilation of 2. Note that when $n = 5$ or 11, tree $IT3_k^h(n)$ can be easily embedded into $CH(n)$ with a dilation of 1.

We conclude this section by mentioning embeddings of other treelike structures that are extensions of incomplete binary trees and that have been investigated in detail in [Boals et al. 1992]. In particular, we have considered n-node incomplete X-trees and leap-trees and have shown that they can be embedded into $CH(n + 1)$ with a dilation of 2. These embeddings are shown to be optimal with respect to dilation. We have also considered the embedding of an n-node binomial tree $B(n)$ into $CH(n)$ and shown that $B(n)$ is a subgraph of $CH(n)$ for say $n \geq 1$; that is, it can be embedded with dilation 1.

4.2. Embedding Meshes

Many useful algorithms, in particular, linear algebra algorithms, can be efficiently performed on meshes with two or more dimensions. Meshes have also been shown to be effective in the solution of partial differential equations. This has led to an interest in simulating meshes on more commonly used multiprocessor architectures, such as hypercubes. Meshes can be embedded with dilation 1 into complete hypercubes if every dimension of the mesh is a power of two. If some of the dimensions are not powers of two, then a dilation of 2 or more is necessary. In fact, Chan [1988] showed that all two-dimensional meshes can be embedded into optimally sized complete hypercubes with dilation 2. The main technique for embedding meshes in complete hypercubes is based on assigning the nodes of each dimension a binary reflected Gray code (BRGC).[1] For example, if a $2^p \times 2^q$ mesh is to be embedded into a Q_{p+q}, then a p-bit BRGC is assigned to the dimension with length 2^p and q-bit BRGC is assigned to the other dimension. The address of a node in a hypercube is obtained by concatenation of the p bits from one dimension with q bits from the other dimension. The minimum-sized hypercube needed to embed a mesh with dilation 1 has been characterized by Havel and Móravek [1972].

Fact 1. *If an* $\alpha_1 \times \alpha_2 \times \ldots \times \alpha_k$ *mesh* M *is embedded in an* n*-dimensional complete hypercube* Q_n *with dilation* 1, *then* $n \geq \Sigma_{i=1}^{k} \lceil \log_2 \alpha_i \rceil$ [Havel and Móravek 1972].

It follows from the above fact that the expansion of the embedding of M into Q_n is in the range of 1 to 2^k. When a dilation-1 embedding is not possible, then dilation-2 and dilation-3 embeddings have been achieved using a variety of techniques including step embedding [Aleliunas and Rosenberg 1982], folding [Leiserson 1980], line compression [Aleliunas and Rosenberg 1982], modified line compression [Chan 1988] and graph decomposition [Ho and Johnsson 1990].

In this section we investigate embeddings of two-dimensional meshes into composite hypercubes. For these embeddings the main difficulty lies in the fact that the number of nodes in a composite hypercube is not a power of two. Moreover, due to the very nature of composite hypercubes, it is interesting to consider embeddings of an n-node mesh into an n-PE composite hypercube; that is, we use exactly the same number of processors. We characterize the class of two-dimensional meshes that can be embedded into a composite hypercube with dilation 1. We also investigate the embeddings of incomplete meshes and composite meshes into composite hypercubes and present dilation-1 embeddings for these architectures.

4.2.1. Embedding Meshes with Dilation 1 and Expansion 1. If both dimensions of a mesh are a power of two, then clearly Gray code embedding is optimal; that is, the embedding using BRGC achieves dilation 1. In this section we show that an n-node two-dimensional mesh can be optimally embedded into composite hypercube $CH(n)$ if one of the dimensions is a power of two. In the case when both dimensions are not a power of two, we show that no dilation-1 embedding exists.

Theorem 5. *Any $\alpha_1 \times \alpha_2$ mesh can be embedded into a $CH(\alpha_1 * \alpha_2)$ with adjacencies preserved if and only if α_1 or α_2 is a power of two.*

Proof: We start by proving the (if) part first. Without loss of generality assume that $\alpha_2 = 2^d$ and $\alpha_1 = 2^{d_1} + 2^{d_2} \ldots + 2^{d_k}$ where $d_1 > d_2 > \ldots > d_k$. We can view an $\alpha_1 \times \alpha_2$ mesh M as a disjoint union of k different meshes $M_1 = 2^{d_1} \times \alpha_2$, $M_2 = 2^{d_2} \times \alpha_2$, $\ldots$, $M_k = 2^{d_k} \times \alpha_2$ with α_2 additional edges between every consecutive pair M_i and M_{i+1}, for $1 \leq i \leq k - 1$. The additional edges between M_i and M_{i+1} result from the adjacency between the last row of M_i and the first row of M_{i+1}. Observe that every row in M_i has α_2 nodes. We call this decomposition of M a k-component decomposition. Clearly each M_i can be embedded with dilation 1 since both dimensions of M_i are powers of two. Our general strategy for embedding M is by assigning a d-bit binary reflected Gray code to the α_2 dimension and by using the first α_1 elements of a $(d_1 + 1)$-bit binary reflected Gray code for the α_1 dimension. An example embedding of a 3×4 mesh into a $CH(12)$ is shown in Figure 8.

We show that the above strategy yields a dilation-1 embedding by using induction on k, the number of components of M. For the base case note that if $\alpha_1 = 1$ or $\alpha_1 = 2^{d_k}$, then clearly an $\alpha_1 \times \alpha_2$ mesh is embeddable into a $CH(\alpha_1 * \alpha_2)$ with dilation 1 since

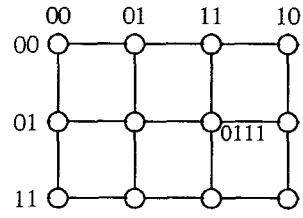


Figure 8. Gray code embedding of a 3×4 mesh.

both dimensions are powers of two. This part of the proof follows by simple induction on the number of submeshes M_i, if one notes that $CH(\alpha_1 * \alpha_2) - CH(2^{d_k})$ is also a composite hypercube.

Now we conclude the proof of the theorem by establishing the fact that if an $\alpha_1 \times \alpha_2$ mesh can be embedded onto a composite hypercube $CH(\alpha_1 * \alpha_2)$ with dilation 1, then either α_1 or α_2 must be a power of two. Let M be an $\alpha_1 \times \alpha_2$ mesh and let $\Phi: M \to CH(\alpha_1 * \alpha_2)$ be an embedding of M onto $CH(\alpha_1 * \alpha_2)$ with dilation 1. By definition of a composite hypercube, $CH(\alpha_1 * \alpha_2)$ can be written as the union of a complete cube Q_d and a composite hypercube $CH(\alpha_1 * \alpha_2 - 2^d)$, where each node of $CH(\alpha_1 * \alpha_2 - 2^d)$ is adjacent to exactly one node in Q_d and each node of Q_d is adjacent to at most one node of $CH(\alpha_1 * \alpha_2 - 2^d)$. We shall call the nodes of $\Phi^{-1}(Q_d)$ red nodes and the nodes of $\Phi^{-1}(CH(\alpha_1 * \alpha_2 - 2^d))$ black nodes. Because of the above-mentioned adjacencies between the nodes of Q_d and the nodes of $CH(\alpha_1 * \alpha_2 - 2^d)$, the set of red nodes and the set of black nodes must consist of stripes (that is, submeshes) with dimension $r \times \alpha_2$ or $\alpha_1 \times r$. Thus $2^d = k * \alpha_2$ or $2^d = \alpha_1 * k$ for some k. Therefore either α_1 or α_2 must be a power of two. ■

We conclude this section by observing that if an expansion of more than 1 is allowed, then all two-dimensional meshes can be embedded with dilation 1. This can be done by first embedding a $\alpha_1 \times \alpha_2$ mesh M into a mesh M' of size $m' = 2^{\lceil \log \alpha_1 \rceil} * \alpha_2$, and then embedding M' into $CH(m')$. Note that the expansion of this embedding is $2^{\lceil \log \alpha_1 \rceil}/\alpha_1$, which is less than 2. In order to keep the expansion minimal, if $2^{\lceil \log \alpha_1 \rceil}/\alpha_1 > 2^{\lceil \log \alpha_2 \rceil}/\alpha_2$ then instead of embedding M into M', we embed M into M'', which is of size $\alpha_1 * 2^{\lceil \log \alpha_2 \rceil}$. This leads to the following corollary.

Corollary 2. *Any $\alpha_1 \times \alpha_2$ mesh can be embedded into a $CH(m)$ with adjacencies preserved, for $m = \min\{\alpha_2 2^{\lceil \log \alpha_1 \rceil}, \alpha_1 2^{\lceil \log \alpha_2 \rceil}\}$.*

Note that our embedding of two-dimensional meshes has a worst-case expansion of less than 2 whereas the embedding presented in [Tzeng et al. 1990] has a worst-case expansion of at most 4.

4.2.2. Embedding Incomplete and Composite Meshes with Dilation 1. In many applications a mesh algorithm may work on a part of the mesh, and in such cases it is interesting to consider meshes that we call *incomplete meshes*. Also it is possible that several mesh-based algorithms are working together, each on a different size of mesh. To capture this idea, we introduce the concept of a *composite mesh*, that is, a mesh consisting of several, smaller different-sized meshes. In this section we show that incomplete and composite meshes can be embedded into composite hypercubes with dilation 1.

We define an incomplete mesh $IM(\alpha_1 \times \alpha_2, \alpha_3)$ to be a $\alpha_1 \times \alpha_2$ dimensional mesh with α_3 rightmost nodes missing in the α_1-th row for $\alpha_3 \leq \alpha_2$ (see Figure 9).

Theorem 6. *An incomplete mesh $IM(\alpha_1 \times \alpha_2, \alpha_3)$, $0 \leq \alpha_3 \leq \alpha_2$ can be embedded into $CH(\alpha_1 * \alpha_2 - \alpha_3)$ with dilation 1, if $\alpha_2 = 2^d$.*

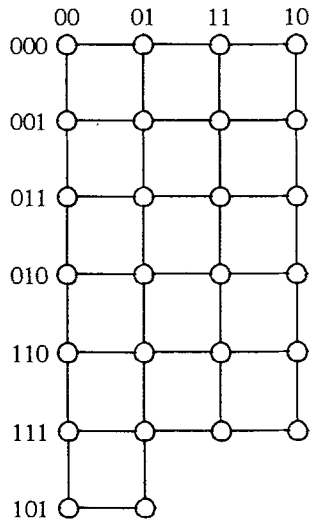


Figure 9. An embedding of $IM(7 \times 4, 2)$ into $CH(26)$.

Proof: We provide an outline of the proof; details are omitted for the sake of brevity. First, note the fact that each row of an $\alpha_1 \times \alpha_2$ mesh with $\alpha_2 = 2^d$ can be embedded into a d-dimensional hypercube with dilation 1, according to the strategy explained in Theorem 5. If the rightmost α_3 nodes are missing, it follows from the definition of the composite hypercube that the last row can be embedded into a $(\alpha_2 - \alpha_3)$-node composite hypercube with dilation 1. The union of this composite hypercube with the $(\alpha_1 - 1) \times \alpha_2$ mesh gives us the desired result. Figure 9 shows an example of an embedding of an $IM(7 \times 4, 2)$ into a $CH(26)$. ■

We define a composite mesh $CM(\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_k)$ as a collection of k meshes M_1, $M_2, \ldots, M_k$ of sizes $\alpha_1 \times \alpha_1, \alpha_2 \times \alpha_2, \ldots, \alpha_k \times \alpha_k$, respectively. In addition, nodes in the first row of M_{i+1} are adjacent to the leftmost nodes in the last row of M_i. More precisely, node $m_i(\alpha_i, r)$ is adjacent to $m_{i+1}(1, r)$ for $r = 1, \ldots, \alpha_{i+1}$ and $1 \leq i < k$, where $m_i(p, q)$ denotes the node in the p-th row and q-th column of mesh M_i.

Theorem 7. *A composite mesh $CM(\alpha_1, \alpha_2, \ldots, \alpha_k)$ can be embedded into a composite hypercube with the same number of nodes, namely $CH(\alpha_1^2 + \alpha_2^2 + \ldots + \alpha_k^2)$, with dilation 1 if every α_i is power of two and $\alpha_i > \alpha_{i+1} > 0$.*

Proof: Since the details are rather straightforward we provide only an outline of the proof. Note that each M_i can be embedded into an optimally sized hypercube Q_{d_i} where $d_i = \log(\alpha_i * \alpha_i)$. That is, each M_i maps to complete subhypercubes in the composite hypercube. Moreover the Gray code embedding ensures that the nodes in the last row of M_i are adjacent to the nodes in the first row M_{i+1} if they exist. This follows from Theorem 6 since we may consider the first row of M_{i+1} along with M_i as an incomplete mesh $IM(\alpha_i \times \alpha_i, \alpha_{i+1})$. Figure 10 shows an example of an embedding of a $CM(4, 2, 1)$ into a $CH(21)$. ■

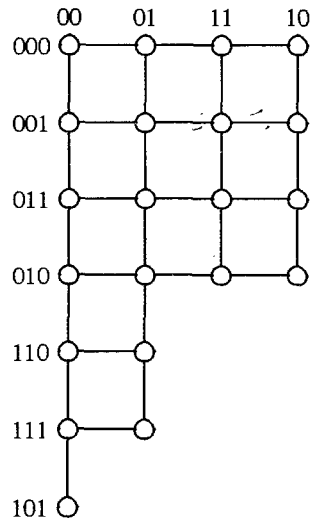


Figure 10. An embedding of $CM(4, 2, 1)$ into $CH(21)$.

4.3. Embeddings of Butterfly and Cube-Connected Cycle Networks

In this section we describe dilation-2 embeddings of a butterfly and a cube-connected cycle network into their optimal-sized composite hypercubes. It is well known that an n-node butterfly $B(n)$ can be embedded into a $2^{\lceil \log n \rceil}$-PE complete hypercube $Q_{\lceil \log n \rceil}$ with a dilation of 1 [Greenberg et al. 1990]. Observe that the expansion of the embedding is greater than 1 but less than 2 provided $n \neq$ a power of two. We show that $B(n)$ can be embedded into an n-node composite hypercube $CH(n)$ with a dilation of 2 and expansion of 1. We also show that this embedding is optimal with respect to dilation for half the values of n. An n-node cube-connected cycle network $CCC(n)$ is known to be embeddable into a $Q_{\lceil \log n \rceil}$ with an optimal dilation of at most 2. In the same vein we show that $CCC(n)$ can also be embedded into an n-node composite hypercube $CH(n)$ with an optimal dilation of at most 2. For the sake of completeness, we next define $B(n)$ and $CCC(n)$.

An n-node butterfly $B(n)$ consists of $h + 1$ levels, with each level containing 2^h nodes. Every node is labeled as a 2-tuple $\langle \alpha, b_1 b_2 \ldots b_h \rangle$, where $1 \leq \alpha \leq h + 1$ and $b_i = 0, 1$ for $1 \leq i \leq h$. A node $v = \langle \alpha, b_1, b_2 \ldots b_h \rangle$ is connected to node $\langle \alpha + 1, b_1 b_2 \ldots b_h \rangle$ by means of a straight edge and v is connected by means of a cross edge to node $\langle \alpha + 1, b_1 \ldots b_{\alpha-1} \bar{b}_\alpha b_{\alpha+1} \ldots b_h \rangle$, for $1 \leq \alpha \leq h$. Thus all the nodes at levels 2 through h have degree 4 and the nodes at levels 1, degree 2. An n-node cube-connected cycle $CCC(n)$ consists of $n = h2^h$ nodes for some positive integer h. Every node in $CCC(n)$ has degree 3 and is labeled as a 2-tuple $\langle \alpha, b_1 b_2 \ldots b_h \rangle$, where $1 \leq \alpha \leq h$ and $b_i = 0, 1$ for $1 \leq i \leq h$. A node $v = \langle \alpha, b_1 b_2 \ldots b_h \rangle$ is connected to three nodes $\langle \alpha - 1 \pmod h, b_1 b_2 \ldots b_h \rangle$, $\langle \alpha + 1 \pmod h, b_1 b_2 \ldots b_h \rangle$, and $\langle \alpha, b_1 \ldots b_{\alpha-1} \bar{b}_\alpha b_{\alpha+1} \ldots b_h \rangle$. Observe that $CCC(n)$ is a symmetric network and the h nodes that have the same second coordinate

form a cycle that we will refer to as the h-cycle. Furthermore, if each of these cycles is collapsed to a node, then we obtain a 2^h-node complete hypercube Q_h.

We now describe an embedding of $B(n)$ into $CH(n)$ that achieves a dilation of 2. Let ϕ be a dilation-1 embedding of a mesh M of size $1 \times (h + 1)$ into $CH(h + 1)$. We know that ϕ exists by Theorem 5. Let $k = \lceil \log(h + 1) \rceil$ and $c_{i,1}c_{i,2} \ldots c_{i,k}$ be the label of the image of the i-th node of M under the embedding ϕ. Now, in order to embed $B(n)$ into $CH(n)$, we assign node $\langle i, b_1b_2 \ldots b_h \rangle$ of $B(n)$ to PE $c_{i,1} \ldots c_{i,k}b_1 \ldots b_h$ of $CH(n)$. Observe that the PEs that get nodes on level i of $B(n)$ assigned form a complete hypercube of 2^h PEs. Furthermore, the PEs that get the nodes in the j-th column of $B(n)$ assigned form a composite hypercube $CH(h + 1)$. By using the fact that a $CH(m2^k)$ can be expressed as a cross product of $Q_k \times CH(m)$, we can easily see that the set of PEs that get the nodes of $B(n)$ assigned by the above embedding procedure form a composite hypercube $CH(n)$. The straight edges of $B(n)$ are mapped to edges of the $CH(n)$ because of the embedding ϕ. The cross edges are mapped to paths of length 2 as follows: Let (v, v') be a cross edge of $B(n)$ between levels i and $i + 1$. The node v is assigned to PE $c_{i,1} \ldots c_{i,k}b_1 \ldots b_i \ldots b_h$ and node v' is assigned to PE $c_{i+1,1} \ldots c_{i+1,k}b_1 \ldots \bar{b}_i \ldots b_h$. We know that $c_{i,j}$s differ from $c_{i+1,j}$s for precisely one value of j under the embedding ϕ. Therefore, the labels of v and v' differ in precisely two bits, namely one in the first k bits and another in the last h bits. This shows that every cross edge of $B(n)$ is dilated by 2 in the embedding of $B(n)$ into $CH(n)$. We thus can state the following result.

Theorem 8. *An n-node butterfly $B(n)$ can be embedded into an n-PE composite hypercube $CH(n)$ with dilation 2.*

The above embedding of $B(n = (h + 1)2^h)$ into $CH(n)$ achieves a dilation of 2, which is optimal whenever h is an even integer. This can be seen from the fact that $B(n)$, in this case, is a connected bipartite graph whose partite sets differ by 2^h nodes, whereas $CH(n)$ is a connected bipartite set with its partite sets of equal size. The question still remains open as to whether the above embedding is optimal with respect to dilation for an odd h. Note that when n is a power of two, $CH(n)$ is a complete hypercube and thus a dilation-1 embedding of $B(n)$ into $CH(n)$ exists [Greenberg et al. 1990].

Finally we outline a dilation-1 embedding of an $n = h2^h$-node cube-connected cycle network into $CH(n)$. Note that $CH(n)$ is isomorphic to the graph $CH(h) \times Q_h$ where $\times$ is the graph product. Since $CH(h)$ is Hamiltonian whenever h is even (see Section 2), the h-cycles of $CCC(n)$ can be embedded into 2^h copies of $CH(h)$ in $CH(h) \times Q_h$ in a parallel fashion so that nodes $\langle \alpha, b_1 \ldots b_\alpha \ldots b_h \rangle$ and $\langle \alpha, b_1 \ldots \bar{b}_\alpha \ldots b_h \rangle$ are assigned to adjacent PEs of $CH(n)$. Note that if h is odd, $CCC(n)$ contains h-cycles of odd length; hence $CCC(n)$ is not isomorphic to any subgraph of the bipartite graph $CH(n)$. Thus we have established the following theorem:

Theorem 9. *A dilation-1 embedding of an $n = h2^h$-node cube-connected cycle network into an n-PE composite hypercube exists if and only if h is even.*

It can also be shown that whenever h is odd we can embed $CCC(n)$ into $CH(n)$ with a dilation of 2.

5. Conclusion

In this paper we have investigated the graph properties of composite hypercubes. We have shown that various types of incomplete binary trees, meshes and butterflies can be optimally simulated by composite hypercubes. In fact, the embedding results indicate that composite hypercubes show simulation capabilities that are very similar to complete hypercubes. Finally, we showed that fully normal algorithms can be efficiently implemented on composite hypercubes with a wide range of architecture sizes.

We have also briefly indicated how to design an efficient algorithm that recognizes composite hypercubes within a complete hypercube. This recognition algorithm is fundamental to task allocation problems and to the development of efficient algorithms on composite hypercubes.

Obviously, our work is simply a start if composite hypercubes are to be shown as an effective parallel interconnection network. There remain a number of issues that need to be investigated for composite hypercubes. For example, simulations of other popular architectures such as shuffle-exchange and pyramids still remain open problems. Design of other classes of parallel algorithms (e.g., normal algorithms) on composite hypercubes also remain to be investigated.

Appendix

Here we define the graph theoretic properties used in Section 2. For more details the reader is referred to standard graph theory text books such as [Bondy and Murthy 1976] or [Behzad et al. 1979].

1. e = number of edges.
2. δ = minimum node degree, and Δ = maximum node degree.
3. κ (resp., λ) = node (resp., edge) connectivity number; that is, the minimum number of nodes (resp., edges) required to disconnect a graph.
4. ω = maximum clique size.
5. χ_0 (resp., χ_1) = node (resp., edge) chromatic number.
6. d = diameter; that is, the distance between two nodes in the graph that are furthest apart. We call $[u, v]$ a *diametrical pair* if the distance between u and v is equal to the diameter of the graph.
7. α_0 (resp., α_1) = node (resp., edge) covering number; that is, the minimum number of nodes (resp., edges) required to cover all the edges (resp., nodes) of a graph.
8. β_0 = node independence number; that is, the size of the maximum set of nodes in a graph such that no two of which share an edge.
9. β_1 = edge independence number (the matching number); that is, the size of the maximum set of edges in a graph such that no two edges in the set are incident on the same node.
10. g = girth; that is, the length of the shortest cycle in the graph.
11. c = circumference; that is, the length of the longest cycle in the graph.
12. θ_0 (resp., θ_1) = node (resp., edge) clique number; that is, the minimum number of cliques that contain all the nodes (resp., edges) of a graph.

Notes

1. A binary reflected Gray code $C(d)$ on d bits is defined as $C(d) = 0C(d-1), 1C^R(d-1)$ with $C(1) = 0, 1$ where $C^R(d-1)$ is a reversal of $C(d-1)$. For example, $C3 = 000, 001, 011, 010, 110, 111, 101, 100$.

References

Aleliunas, R., and Rosenberg, A. 1982. On embedding rectangular grids into square grids. *IEEE Trans. Comps.*, C-31: 907–913.

Behzad, M., Chartrand, G., and Lesniak-Foster, L. 1979. *Graphs and Diagraphs*. Prindle, Weber, and Schmidt, Boston.

Bhatt, S.N., and Ipsen, I.C.F. 1985. How to embed trees in hypercubes. Res. rept. 443 (Dec.), Dept. of Comp. Sci., Yale Univ., New Haven, Conn.

Bhatt, S., Chung, F., Leighton, F.T., and Rosenberg, A. 1986. Optimal simulations of tree machines. In *Proc., 27th Symp. on the Foundations of Comp. Sci.*, pp. 274–282.

Boals, A., Gupta, A., and Sherwani, N. 1992. Incomplete hypercubes: Embeddings and algorithms. Tech. rept. TR/92-22, Dept. of Comp. Sci., W. Mich. Univ., Kalamazoo, Mich.

Bondy, J.A., and Murthy, U.S.R. 1976. *Graph Theory with Applications*. MacMillan, New York.

Chan, M.Y. 1988. Dilation-2 embeddings of grids into hypercubes. Tech. rept. UTDCS 1-88, Dept. of Comp. Sci., Univ. of Tex. at Dallas.

Chan, M.Y., and Lee, S.-J. 1993. Fault-tolerant permutation routing in hypercubes. *J. Parallel and Distr. Comp.* (Apr.): 227–281.

Chen, H.L., and Tzeng, N.-F. 1989. Enhanced incomplete hypercubes. In *Proc., Internat. Conf. on Parallel Processing*, vol. 1, pp. 270–277.

Chen, M., and Shin, K.G. 1987. Processor allocation in a N-CUBE multiprocessor using Gray codes. *IEEE Trans. Comps.*, C-36, 12: 1396–1407.

Chen, M., and Shin, K.G. 1988. Message routing in an injured hypercube. In *Proc., 3rd Conf. on Hypercube Concurrent Comps. and Applications*, pp. 312–317.

Das, S.K., Deo, N., and Prasad, S. 1990. Parallel graph algorithms for hypercube computers. *Parallel Computing*, 13: 143–158.

Fishburn, J.P., and Finkel, R.A. 1982. Quotient networks. *IEEE Trans. Comps.*, C-31, 4: 288–295.

Gordon, J.M., and Stout, Q.F. 1988. Hypercube message routing in the presence of faults. In *Proc., 3rd Conf. on Hypercube Concurrent Comps. and Applications*, pp. 318–327.

Graham, N., Harary, F., Livingston, M., and Stout, Q.F. 1993. Subcube fault-tolerance in hypercubes. *Information and Computation* (Feb.): 218–314.

Greenberg, D.S., Heath, L.S., and Rosenberg, A.L. 1990. Optimal embeddings of butterfly-like graphs in the hypercubes. *Math Systems Theory*, pp. 61–77.

Gupta, A.K. 1989. On the relationship between parallel computation and graph embeddings. Ph.D. thesis, Purdue Univ., Lafayette, Ind.

Harary, F., Hayes, J.P., and Wu, H.-J. 1988. A survey of the theory of hypercube graphs. *Computational Math. and Applications*, 15, 4: 277–289.

Hastad, J., Leighton, F.T., and Newman, M. 1987. Reconfiguring a hypercube in the presence of faults. In *Proc., 19th Annual ACM Symp. on the Theory of Computing*, pp. 274–284.

Hastad, J., Leighton, T., and Newman, M. 1989. Fast computation using faulty hypercubes. In *Proc., 21st Annual Symp. on the Theory of Computing*, pp. 251–263.

Havel, I., and Móravek, J. 1972. B-valuations of graphs. *Czech Math. J.*, 22: 338–351.

Ho, C.T., and Johnsson, S.L. 1990. Embedding meshes in Boolean cubes by graph decomposition. *J. Parallel and Distr. Comp.*, 8: 325–339.

Hong, J.W., Mehlhorn, K., and Rosenberg, A. 1983. Cost trade-offs in graph embeddings, with applications. *JACM*: 709–728.

Kandlur, D.D., and Shin, K.G. 1988. Hypercube management in the presence of node failures. In *Proc., 3rd Conf. on Hypercube Concurrent Comps. and Applications*, pp. 328–336.

Katseff, H.P. 1988. Incomplete hypercubes. *IEEE Trans. Comps.*, 37, 5: 604–608.

Kosaraju, S.R., and Atallah, M. 1988. Optimal simulations between mesh connected array of processors. *JACM*, 35, 3: 635–650.

Lee, T.C., and Hayes, J.P. 1988. Routing and broadcasting in faulty hypercube computers. In *Proc., 3rd Conf. on Hypercube Concurrent Comps. and Applications*, pp. 346–354.

Leiserson, C.E. 1980. Area-efficient graph layouts (for VLSI). In *Proc., 22nd Annual IEEE Symp. on the Foundations of Comp. Sci.*, pp. 270–281.

Prabhalla, V., and Sherwani, N. 1990. Parallel single row routing on compact hypercubes. Tech. rept. TR/90-06, Dept. of Comp. Sci., W. Mich. Univ., Kalamazoo, Mich.

Roy, A., Deogun, J.S., and Sherwani, N.A. 1989. A parallel algorithm for single row routing problems. *J. Circuits, Systems, and Comps.* (to appear).

Saad, Y., and Schultz, M.H. 1988. Topological properties of hypercubes. *IEEE Trans. Comps.*, 37, 7: 867–872.

Tien, J.-Y., and Yang, W.-P. 1991. Hierarchical spanning trees and distributing on incomplete hypercubes. *Parallel Comp.*, 17: 1343–1360.

Tzeng, N.-F. 1990. Structural properties of incomplete hypercubes. In *Proc., 10th Internat. Conf. on Distr. Computing Systems*, pp. 262–269.

Tzeng, N.-F., Chen, H.L., and Chuang, P.J. 1990. Embeddings in incomplete hypercubes. In *Proc., Internat. Conf. on Parallel Processing*, pp. III-335–III-339.