Yale University
Department of Computer Science

Routing Multiple Paths in Hypercubes

David S. Greenberg   Sandeep N. Bhatt*

Department of Computer Science
Yale University
New Haven, CT 06510

YALEU/DCS/TR-768

*Currently visiting the Computer Science Department, 256–80 California Institute of Technology, Pasadena, CA 91125.

Abstract

We present new techniques for mapping computations onto hypercubes. Our methods speed up classical implementations of grid and tree communications by a factor of $\Theta(n)$, where n is the number of hypercube dimensions. The speed-ups are the best possible.

We obtain these speed-ups by mapping each edge of the guest graph onto short, edge-disjoint paths in the hypercube. These multiple–path embeddings can be used to reduce communication time for large grid–based scientific computations, to increase tolerance to link faults, and for fast routing of large messages.

We also develop a general technique for deriving multiple–path embeddings from multiple–copy embeddings. Multiple–copy embeddings are one-to-one maps of independent copies of the guest graph within the hypercube. We present an efficient multiple–copy embedding of the cube-connected-cycles network within the hypercube. This embedding is used to derive efficient multiple–path embeddings of trees and butterfly networks in hypercubes.
Key words: Graph embedding, hypercube, multi-dimensional grids, binary trees, FFT, multiple–path embedding, multiple–copy embedding.

1 Introduction

Standard embeddings [5, 7, 11, 12] of computational structures into hypercube networks [13, 25] suffer one weakness: they use only a tiny fraction of the available communication links. We present techniques to utilize all the communication links at every step. Besides speeding up the transfer of large amounts of data, our techniques can also be used to incorporate fault-tolerant transmission schemes and to facilitate bit-serial, wormhole routing [1].

Consider the classical binary reflected gray code mapping of cycles in hypercubes. Each edge of a directed cycle is mapped to a unique hypercube link. Of the n directed links out of each hypercube node, exactly one is used, and the remaining $n-1$ are idle. If each node is to send m packets to its successor along the cycle then $m/2$ communication steps are necessary (see Section 2).

We present new techniques to reduce the communication time to $\Theta(m/n)$. The resulting $\Theta(n)$ speedup over classical techniques is the best asymptotically possible. Our techniques also speed up communication for multi-dimensional grids, FFTs, and binary trees. These structures arise frequently in scientific and signal processing applications.

Our main technical tool is the notion of a *multiple–path embedding*, one which maps each edge of the computation graph onto edge–disjoint paths in the connection network. The multiple paths can be used to increase the effective throughput. Alternatively, if communication links are unreliable multiple paths can be used to increase fault-tolerance. For example, Rabin's IDA scheme [22] can be implemented along the independent paths.

Multiple-path embeddings also lend insight into the assembly of large systems. The assembly of large systems is severely constrained by the number of pin connections available per node (chip or board). With W pins per node, there is a simple tradeoff in the choice of network between the number of communication channels and their width. For example, N nodes may be interconnected as a hypercube with $O(N \log N)$ channels, each of width $O(W/\log N)$. Alternatively, they may be connected as a two-dimensional grid with $O(N)$ channels, each of width $O(W)$. This constant pinout model is analogous for multi-chip assemblies to the constant wire-density model of Dally and Seitz [9] for networks laid out on a single chip or wafer.

One might suspect that a network designed for one particular communication pattern would outperform a more general interconnection using narrower channels. Our multiple–path embedding results show that this need not be true; the *narrow* hypercube can simulate the *wide* grid with $O(1)$ slowdown (assuming unit delay on all wires). At the same time the narrow hypercube retains the flexibility to service low diameter patterns such as binary trees or FFTs considerably faster than the wide grid.

We also investigate two other methods of using the hypercube edges efficiently, *multiple–copy embeddings* and *large–copy embeddings.* In a multiple–copy embedding several independent copies of a computation graph are embedded one-to-one into the hypercube. Both Ho and Johnsson [14] and Stout and Wager [26] have studied multiple–copy embeddings of the spanning binomial tree. We present multiple–copy embeddings of cube-connected cycles (CCCs) and FFTs. We then develop a general method for transforming a multiple–copy embedding of one graph into a multiple–path embedding of another.

The remainder of this paper is organized as follows. Section 2 illustrates one use of multiple–path embeddings. Section 3 reviews basic definitions. Section 4 presents multiple–path embeddings of grids. Multiple–copy embeddings of CCC networks are given in Section 5. Section 6 derives a general framework for converting multiple–copy embeddings into multiple–path embeddings. The general framework is used to create multiple–path embeddings of complete-binary trees. Section 7 explores applications of the general technique to bit-serial routing. Section 8 discusses large–copy embeddings and compares multiple–path, multiple–copy, and large–copy embeddings. Section 9 concludes with some unresolved problems.

2 An Illustration

Figure 1 shows the classical binary reflected gray code embedding of the directed cycle in the hypercube. The label on an edge (u, v) corresponds to the dimension of the image of (u, v) in the hypercube. In one step the image of u can send one packet to the image of v along the image of (u, v). However, when u has m packets to send, they must be sent sequentially, requiring m steps. A more efficient method would use the idle edges incident to the image of u to allow packets to be sent concurrently along multiple paths.

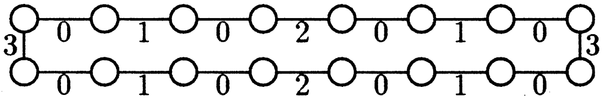

Figure 1: The binary reflected graycode embedding

Unfortunately, with the classical gray code the idle edges cannot be employed to speed up transmission. To see this, consider the total number of packets whose paths must cross dimension 0, the most frequently used dimension. There are $m2^{n-1}$ such packets, but only 2^n directed edges in dimension 0. Since only one packet can cross an edge in one step, $m/2$ steps are necessary.

Section 4 gives an embedding which allows $n/2$ packets from each node to be sent in just 3 steps. To avoid congestion our embedding uses all dimensions uniformly and, furthermore, maps each cycle edge to $n/2$ edge-disjoint paths of length 3. Thus with m packets per node, our strategy requires $\Theta(m/n)$ steps, the best achievable. In fact, by mapping a cycle of length 2^{n+1} optimal throughput is achieved.

Grid relaxation methods for solving differential equations can be speeded by using multiple–path embeddings. Consider a large two-dimensional grid relaxation performed on a hypercube. Suppose that the grid has dimensions $M \times M$ while the hypercube has $N^2 < M^2$ nodes. The grid is partioned into $M/N \times M/N$ subgrids, with one subgrid mapped to each hypercube node. This evenly balances the load and minimizes the total communication. Each node has to communicate M/N packets from the perimeter of its subgrid to each of the four nodes containing adjacent subgrids.

The multiple–path embedding guarantees that the communication in each relaxation step can be performed in $\Theta(M/(N \log N))$ steps, instead of the $\Theta(M/N)$ steps required by the traditional gray code methods.

3 Preliminaries

This section presents basic definitions used in the remainder of this paper.

Graph Embeddings

An *embedding* of a guest graph, $G = (V, E)$, into another graph, the host graph, $H = (W, F)$, is a one-to-one mapping $\eta : V \rightarrow W$ along with a map μ which assigns each edge $(u, v) \in E$ to a path in H from $\eta(u)$ to $\eta(v)$. When $|V| > |W|$ we allow many-to-one mappings, but restrict the mapping so that each host vertex is the image of no more than $\lceil |V|/|W| \rceil$ guest vertices. The quantity $\lceil |V|/|W| \rceil$ is called the *load* of the embedding.

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 G.

The *congestion* of an edge $f \in F$ equals the number of edges in G whose images contain f. The congestion of an embedding is the maximum congestion of any edge in H.

Throughout this paper the host graph models a parallel computer — its vertices represent the processors and its edges represent communication links between processors. Similarly a guest graph represents a parallel computation — its vertices represent processes and its edges connect processes which must communicate. We assume that the guest, or communication, graph does not change over time. In one phase of the computation each process sends a message to each of its neighbors in the communication graph. Furthermore, during one time unit in the network, each processor can send one message packet over each outgoing link.

The *one-packet cost* of an embedding of G into H is the number of time units necessary for H to complete one phase of G when each message contains one packet. It is easily seen that the cost is no less than the maximum of the dilation and the congestion. Similarly, the one-packet cost is no greater than the product of the dilation and the congestion. Leighton, Maggs, and Rao [19] show that the one-packet cost can be reduced to no more than a constant factor times the sum of the dilation and the congestion.

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. When packets can be pipelined along the paths in H or if there are multiple paths in H per edge of G then the p–packet cost can be less than p times the one-packet cost.

Multiple–copy embeddings

A *k–copy embedding* of G into H is a collection of k one-to-one embeddings of G into H. Since each embedding is one-to-one, each node of H can host up to k nodes, one from each copy of G. The congestion of an edge $f \in H$ in a k–copy embedding equals the sum, over all embeddings, of the congestion on f. The *edge-congestion* of a k–copy embedding is the maximum congestion on any edge in H.

Multiple–path embeddings

A *width–w* embedding of G into H is a one-to-one embedding in which each edge of G is mapped to w edge-disjoint paths in H. The congestion of an edge $f \in H$ equals the number of edges $e \in G$, one of whose image paths contains f. The congestion of the embedding equals the maximum congestion among all edges in H.

Boolean Hypercubes and Graycodes

The *k–dimension hypercube*, Q_k, consists of 2^k nodes with distinct k–bit addresses. There is a directed edge (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 ith bit is said to lie in the ith dimension.*

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

We let H_k denote the following sequence of nodes of Q_k: $H_k(0) = 0^k$, and $H_k(i+1)$ is the neighbor of $H_k(i)$ across dimension $G_k(i)$. It is well-known that H_k is a hamiltonian cycle [24].

Cross Products

The *cross-product* of two graphs $G = (V, E)$ and $H = (W, F)$ is denoted $G \times H$ and consists of vertex set $V \times W = \{\langle v, w\rangle \mid v \in V, w \in W\}$ and edge set $\{(\langle v, w_1\rangle, \langle v, w_2\rangle) \mid v \in V, w_1 \in W, w_2 \in W\} \cup \{(\langle v_1, w\rangle, \langle v_2, w\rangle) \mid v_1 \in V, v_2 \in V, w \in W\}$. 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 (resp. cycle) and the length W path (resp. cycle) is the $L \times W$ grid (resp. torus). Similarly, the cross product $Q_n \times Q_m$ is equal to Q_{n+m}.

3.1 Multiple–Copy Embeddings of Cycles in Hypercubes

Hypercubes can be decomposed into edge-disjoint hamiltonian cycles (see [3] for a survey). In particular, Alspach, Bermond, and Sotteau [3] show that the edges of every (undirected) hypercube with $2n$ dimensions can be partitioned into n (undirected) hamiltonian cycles. Furthermore, if the number of dimensions is $2n+1$, then the edges can be partitioned into n cycles and one perfect matching.

These results are easily extended into multiple–copy embeddings of directed cycles in directed hypercubes. For an n-dimensional cube, we orient each of the $\lfloor n/2 \rfloor$ undirected cycles in either direction to obtain the following lemma. When n is odd it is not, in general, possible to partition Q_n into n directed cycles.

Lemma 1 *For n even (odd), n $(n-1)$ copies of the 2^n-node directed cycle can be embedded into Q_n with dilation 1 and congestion 1.*

*We define the hypercube as a directed graph, thus each communication link is modeled as a directed edge.

3.2 Moments

We now introduce the notion of the *moment* of a node in the hypercube. The moment of a node in Q_n is an $\log n$–bit label which has the property that all the neighbors of a node have distinct labels. This simple property underlies all the multiple–path embeddings presented in this paper. In the following, $b(x)$, $0 \leq x < n$ denotes the $\log n$–bit binary representation of the number x. Also, $\bigoplus$ denotes the *bitwise* xor of $\log n$–bit numbers.

Definition 1 *The* moment *of an n–bit number $v = v_{n-1}, v_{n-2}, \ldots, v_0$ is defined by $M(0) = b(0)$ and $M(v) = \bigoplus_{i|v_i=1} b(i)$.*

Lemma 2 *Each hypercube neighbor of a given node, u, has a distinct moment.*

Proof: Let v and w be the neighbors of u in dimension i and j. Then $M(v) = M(u) \oplus b(i) \neq M(u) \oplus b(j) = M(w)$.∎

4 Multiple–path Embeddings of Grids

In this section we show that grids have efficient multiple–path embeddings in hypercubes. We present the technique for cycles; the extension to multi-dimensional grids is noted at the end of the section.

4.1 Multiple path embeddings of cycles

We present two multiple–path embeddings of cycles. The first embedding maps the 2^n–node cycle into Q_n with load 1, width $\lfloor n/2 \rfloor$ and cost 3. Roughly speaking, half of all hypercube edges transmit a packet at each of the 3 steps. The second embedding attempts to keep all hypercube edges busy at each step; it maps the 2^{n+1}–node cycle into Q_n with load 2, width $\lfloor n/2 \rfloor$, and cost 3.

A key idea in both multiple–path embeddings of cycles will be the conversion of the multiple–copy embedding of Lemma 1 into a multiple–path embedding. In Section 6 this idea will be generalized to allow the construction of additional multiple–path embeddings.

4.2 Embedding Cycles with Load 1

Theorem 1 *The length-2^n directed cycle can be embedded in Q_n with width $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$-packet cost 3.*

Proof: Suppose that $n = 4k + r$, $0 \leq r < 4$, $k > 0$.[†] Thus for $r = 0, 1$ we have $\lfloor n/2 \rfloor = 2k$ and for $r = 2, 3$ we have $\lfloor n/2 \rfloor = 2k + 1$.

First we partition Q_n as the product $Q_{2k} \times Q_{2k+r}$. The product can be visualized as a grid with 2^{2k} rows and 2^{2k+r} columns. Each row is connected as Q_{2k+r} and each column is connected as Q_{2k}. The most significant $2k$ bits of the addresses in Q_n name a grid row while the least significant $2k + r$ bits name a grid column.

Furthermore we partition the columns into 2^r blocks by letting the least significant r bits of the column name be the name of a block and the most significant $2k$ bits be the name of a position within a block. Note that within each block each row and column is connected as Q_{2k}. Thus if each column is treated as a *coarse* node it has $2k$ neighboring columns within the block. (See Figure 2)

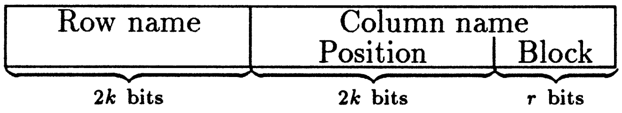

Figure 2: Dividing addresses into three fields

Next we number the k edge-disjoint undirected cycles and the $2k$ edge-disjoint directed cycles of Q_{2k} provided by Lemma 1. The undirected cycles are numbered arbitrarily. Then for $0 \leq i < k$ let directed cycle $2i$ be the ith undirected cycle oriented in one direction and directed cycle $2i + 1$ be the ith undirected cycle oriented in the other direction.

Now we can choose a *special* cycle within the subcube associated with each column. For column c at position x in block b we select the edge-disjoint directed cycle number $M(x)$[‡] as the special cycle. Observe that each node of Q_n lies in exactly one special cycle, and we have selected 2^{2k+r} special cycles.

We now use these special cycles, plus a few edges in the rows to form our length-2^n cycle, C. (See Figure 3.) The cycle C consists of $2^k - 1$ consecutive edges from

[†] $k = 0$ is trivial

[‡] Recall that $M(x)$ denotes the moment of x.

each special cycle and 2^{2k+r} row edges; each row edge connects one column's special cycle to the special cycle in the next column. The order in which $\mathcal{C}$ visits columns is specified by the gray code G_{2k+r} defined in Section 3.

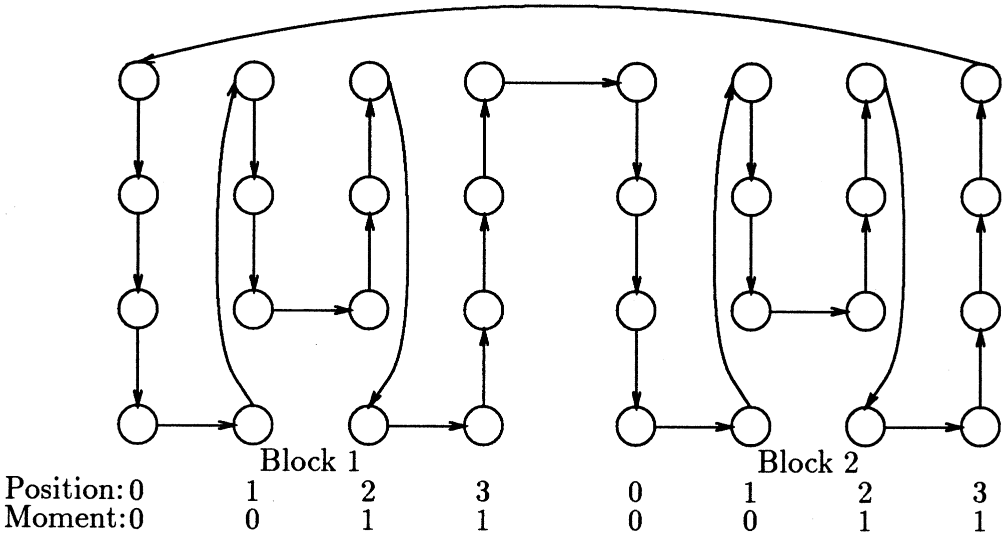

Figure 3: Forming the length-2^n cycle, $\mathcal{C}$, from column special cycles

Formally, the first vertex of $\mathcal{C}$ is the hypercube node at row 0 of column 0. The first $2^k - 1$ edges of $\mathcal{C}$ follow the special cycle in column 0 (until the special cycle is about to return to row 0). The next edge of $\mathcal{C}$ is the row edge in the first dimension of G_{2k+r}. In the new column reached, and each successive column thereafter, $\mathcal{C}$ follows $2^k - 1$ edges of the special cycle and then leaves via the row edge in the next dimension listed in G_{2k+r}. Upon returning to column 0 the cycle $\mathcal{C}$ is complete.

By construction the dimensions of G_{2k+r} will cause $\mathcal{C}$ to visit all 2^{2k+r} columns exactly once and the traversal of special cycles will ensure that each node in each column is visited once. It remains to show that when $\mathcal{C}$ returns to column 0 it is in row 0.

Starting with column 0 group the columns, in the order they were visited, into sets of four. Between columns within each set the dimensions specified by G_{2k+r} for row edges are always 0, 1, and 0. Thus within each set the moments of the first two columns are the same ($x \oplus 0 = x$) as are the moments of the last two. Furthermore the cycle associated with the moment of the first two columns is the reverse orientation of the cycle associated the moment of the last two. (The names of the cycles were chosen so that names differing in the least significant bit corresponded to opposite orientations of the same undirected cycle.) Thus the path taken by $\mathcal{C}$ in the first two columns is reversed in the next two columns thereby returning $\mathcal{C}$ to row 0. Since the number of columns is divisible by four $\mathcal{C}$ must end in row 0 after visiting all the columns.

We are now ready to make the edges of $\mathcal{C}$ wide. Our main tool will be the following observation about the special cycles. Consider column c in block b, and its $2k$ neighboring columns within block b. By Lemma 2 each neighbor has a distinct moment and thus a distinct special cycle. Therefore when the special cycles of all the neighbors are projected onto column c their images are edge disjoint. We use this property by replacing each special edge with length-three paths which cross into neighboring columns, follow the projection of the edge in the neighboring column, and then cross back into the original column. (See Figure 4.)

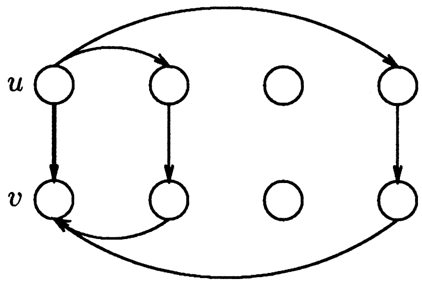

Figure 4: The length-three paths

Formally, we choose the first $2k$ edge-disjoint paths for edge (u, v) along a special cycle (suppose u and v differ in dimension i, $2k + r \leq i < 4k + r$) as follows. The jth path, $0 \leq j < 2k$, from u to v is of the form $u, u \oplus 2^{r+j}, u \oplus 2^{r+j} \oplus 2^i, v$. In other words the jth path crosses into the column adjacent via the edge in dimension $r + j$ (one of the dimensions used within block row subcubes), crosses the ith dimension while remaining in this new column, and then crosses back to its original column via an edge in dimension $r+j$.

We add a $2k+1$st path of length one which goes directly from u to v via the edge in dimension i.

For the edges in the rows we similarly pick $2k+1$ edge-disjoint paths. The first $2k$ paths for row edge (u,v) (where the dimension i between u and v is now such that $0 \leq i < 2k+r$) are chosen so that path j, $0 \leq j < 2k$, is of the form $u, u \oplus 2^{2k+r+j}, u \oplus 2^{2k+r+j} \oplus 2^i, v$. Again we add a $2k+1$st path which goes directly across dimension i.

We claim that this multiple-path embedding of the special cycles has cost 3. First observe that each first edge of the paths corresponding to a single edge of $\mathcal{C}$ is in a different dimension. Thus all first edges emanating from each node are disjoint. Similarly, the set of final edges is also disjoint. Next, we argue that the middle edges are disjoint from one another.

All the middle edges of paths for column edges are projections of special cycle edges onto neighboring columns within a block. But, as we observed earlier, the projections of all special cycles onto any column are edge-disjoint. Therefore these middle edges are disjoint from one another. The middle edges of paths for row edges are projections into neighboring rows and thus disjoint from the column path's middle edges. Furthermore each row edge in $\mathcal{C}$, and all its projections, connects a unique pair of columns and thus its projections are disjoint from those of any other row edge in $\mathcal{C}$. Thus all the middle edges of all the paths are disjoint.

Thus a packet may be sent along all paths including the direct path on step one and forwarded along all the length-three paths on steps two and three. Furthermore an additional packet can be sent along the direct path on step three. Thus the embedding yields $(2k+2)$-packet cost 3 which is always better than that required by the theorem.∎

Since this first embedding only uses only about half the hypercube edges a natural question is whether the idle edges can also be put to use. We do not know how to do so by increasing the width. However, by increasing the load to two and thereby doubling the number of guest (cycle) edges the embedding in the next section uses nearly all the hypercube edges.

4.3 Load 2 Embeddings which Fully Utilize the Hypercube Links

Theorem 2 *The length-2^{n+1} directed cycle can be embedded in Q_n with width $w(n)$ and $w(n)$-packet cost $c(n)$, where:*

For $n \equiv 0, 1 \pmod 4$	$w(n) = \lfloor n/2 \rfloor$	and	$c(n) = 3$.	
For $n \equiv 2, 3 \pmod 4$	$w(n) = \lfloor n/2 \rfloor - 1$	and	$c(n) = 3$,	or
	$w(n) = \lfloor n/2 \rfloor$	and	$c(n) = 4$.	

Proof: Of the four cases, $n \equiv 3 \pmod 4$ requires the most general argument. Thus we start with a proof for the case $n = 4k + 3$, $w(n) = \lfloor n/2 \rfloor - 1$, and $c(n) = 3$. The remaining (simpler) cases will be discussed at the end.

As in the proof of Theorem 1 partition Q_n as the product $Q_{2k} \times Q_{2k+3}$, by dividing the address of each node into two parts. Also partition the 2^{2k+3} columns into eight blocks by letting the least significant three bits of the column name be the name of a block and the the most significant $2k$ bits be the name of a position within a block.

The product can be visualized as a 2^{2k} by 2^{2k+3} grid of nodes with each row connected as Q_{2k+3} and each column connected as Q_{2k}. The most significant $2k$ bits of the addresses in Q_n name a grid row, the next $2k$ bits name a grid column's position within a block and the least significant 3 bits name the column's block. The rows and columns within a block each form a subcube of dimension $2k$ as do the columns when considered as *coarse* nodes.

We number $2k$ of the edge-disjoint directed cycles of both Q_{2k} and Q_{2k+3} arbitrarily. Within each column in position x we choose cycle $M(x)$ of Q_{2k} as the special cycle while in row r we choose cycle $M(r)$ of Q_{2k+3}. Thus each node of Q_n is on two special cycles and the subgraph of Q_n induced by the special cycles spans Q_n and has indegree and outdegree equal to 2 at every node.

By choosing the Eulerian tour of the induced spanning graph as our length-2^{n+1} cycle we need only show that each edge of the special cycles can be given width $w(n)$ with cost $c(n)$.

Each special cycle is given width $2k$ by choosing $2k$ edge-disjoint paths in the following manner. If edge (u, v) is along a column special cycle (and thus u differs from v in dimension i, $2k + 3 \leq i < 4k + 3$) then the jth path, $0 \leq j < 2k$, from u to v is of the form $u, u \oplus 2^{3+j}, u \oplus 2^{3+j} \oplus 2^i, v$. On the other hand if edge (u, v) is along a row cycle (and thus traverses dimension i, $0 \leq i < 2k + 3$) then the jth path,

$0 \leq j < 2k$, from u to v is of the form $u, u \oplus 2^{3+2k+j}, u \oplus 2^{3+2k+j} \oplus 2^i, v$. (Note that we could not add the direct path as a $2k+1$st path because for columns edges the direct edge is used by row paths and vice versa.)

We claim that this width–$2k$ embedding of the special cycles has $2k$-packet cost 3. Observe that the set of first edges on all the length 3 paths for column edges are in dimensions less than $2k+3$ while the set of first edges on paths for row edges are in dimensions greater than or equal to $2k+3$. Thus the first edges emanating from each node, and therefore all the first edges, are disjoint. Similarly, the final edges are also disjoint. Next, we argue that the middle edges are disjoint.

All the middle edges of paths for column edges are projections of special cycle edges onto neighboring columns within a block. But, as we observed earlier, the projections of all special cycles onto any column are edge-disjoint. Therefore these middle edges are disjoint from one another. Similarly middle edges of paths for row edges are projections onto neighboring rows and are disjoint. Since the row edges are disjoint from the column edges all the middle edges are disjoint.

Since the sets of first, middle, and last edges are each disjoint packets can be sent down each path in three steps without interference and the $2k$-packet cost is indeed 3. Having concluded the proof for $n = 4k+3$, $w(n) = \lfloor n/2 \rfloor - 1 = 2k$, and $c(n) = 3$ we now sketch how for $n = 4k+3$ to achieve $w(n) = \lfloor n/2 \rfloor = 2k+1$ with $c(n) = 4$ and then how to handle other values of n.

To obtain width $\lfloor n/2 \rfloor = 2k+1$ we partition Q_{4k+3} into $Q_{2k+1} \times Q_{2k+2}$. Since every cycle must have width $2k+1$, we must have $2k+1$ special cycles within the rows and the columns. The rows have $2k+2$ disjoint cycles, so any subset of $2k+1$ cycles can be chosen as the special cycles. However, the columns have only $2k$ disjoint cycles (by Lemma 1) so one cycle must be chosen twice as a special cycle. This adds one to the congestion on middle edges, and to the cost as well. Therefore, the $\lfloor n/2 \rfloor$–packet cost is 4. (Note that if each node sent $2k$ batches of $2k+1$ packets and a different edge-disjoint cycle were used twice in each batch then the $2k(2k+1)$-packet cost would be $3(2k)+1$ and not $4(2k)$).

The other cases for n are proved in almost identical manner. The cases $n \equiv 0, 1 \pmod 4$ do not require a cycle to be used twice since $\lfloor n/2 \rfloor = 2k$. In the case $n \equiv 2 \pmod 4$, cost 4 and width $2k+1$ is obtained by partitioning $Q_{4k+2} = Q_{2k+1} \times Q_{2k+1}$. The partition into $Q_{2k} \times Q_{2k+2}$ yields cost 3 and width $2k$. When $n \equiv 0 \pmod 4$ all the hypercube edges are in use during each of the 3 steps. ∎

4.4 Bounds on Width and Cost

We next show that the multiple-path embedding of Theorem 2 is the best possible for the cases $n \equiv 0, 1 \pmod 4$.

Lemma 3 *For $w > 2$, every width–w embedding requires dilation (and therefore p-cost, $p \geq w$) at least 3. There is no p-packet cost 3 embeddings of length–2^{n+1} cycles in Q_n with $p > \lfloor n/2 \rfloor$.*

Proof: In order to have $w > 2$ edge-disjoint paths between two distinct hypercube nodes, one of the paths must have length 3 or greater. Therefore, for all width–w embeddings, $w > 2$, the cost must be 3 or greater.

The number of edges traversed by all the paths in the embedding equals the sum, over all edges, of their dilation. Since, to achieve cost 3, at least $w-1$ edges must have dilation 3 this sum is greater than $2^{n+1} \times (w-1) \times 3$. On the other hand the number of hypercube edges available during three steps is simply three times the number of directed hypercube edges or $3n2^n$. Thus in order for the number of edges used to be no greater than the number of edges available, we have that $6 \cdot 2^n(w-1) < 3n2^n$ which implies that $w \leq \lfloor n/2 \rfloor$.∎

4.5 Multiple–path embeddings of grids

In Section 3 we mentioned that grids/tori are cross-products of paths/cycles and that hypercubes are cross-products of smaller hypercubes. These two facts lead to a natural technique for extending embeddings of paths into embeddings of grids. Each axis of the grid is embedded in a hypercube via the cycle embedding and then the cross-product of the hypercubes inherits the embeddings of the axes and thus an embedding of the grid.

For example, when the k-dimensional grid with each side of length 2^a is embedded in Q_{ak} using the cross-product decomposition and the embedding of Theorem 1, we obtain a width–$\lfloor a/2 \rfloor$ embedding with $\lceil a/2 \rceil$–packet cost 3.

When the sides of the grid are equal, but not a power of 2, the cross-product embedding may use the hypercube nodes inefficiently. For example the 5 by 5 grid is the cross-product of two 5 node paths. Each 5 node path can be embedded in an 8 node hypercube and the cross-product of two 8 node hypercubes is a 64 node hypercube. Thus by embedding each axis into its own independent factor subcube we obtain an embedding of the 5 by 5 grid into a 64 node hypercube. The 25 grid nodes could, however, have fit in a 32 node hypercube. The ratio of the size of the hypercube used to the size of the smallest hypercube at least as large as the guest graph is often called the *expansion*. In the 5×5 grid example the expansion is 2; in general cross-product decomposition can lead to expansion $k+1$ for k-axis grids.

Corollary 1 *The k–axis grid with all side lengths equal to L can be embedded in $Q_{k\lceil \log L \rceil}$ with expansion $k+1$, width $\lfloor \lceil \log L \rceil /2 \rfloor$ and $\lceil \lceil \log L \rceil /2 \rceil$-packet cost 3.*

Proof: : Embed each axis in $Q_{\lceil \log L \rceil}$ via the embedding of Theorem 1 and use the cross-product decomposition.∎

For width–1 embeddings it has been shown that using gray codes for the paths and applying the cross-product technique to k–axis grids causes expansion no greater than $k+1$. Chan [8] has shown that by abandoning the cross product approach and increasing the dilation to $O(k)$ the expansion can be reduced to one.

Unfortunately Chan's techniques do not apply immediately to our multiple–path embeddings. Thus we have no current alternative to the cross-product technique.

When the sides of the grid are not equal (more precisely when the ceiling of the logarithm of their lengths are not equal) the multiple–path embeddings require additional work. We cannot just embed each axis in a hypercube large enough to hold it because the width of the embedding depends on the number of dimensions in the host hypercube. If one axis of a grid were embedded to a smaller hypercube than another axis then the width of edges on the first axis would be smaller than the width for edges on the other.

In order to compensate for the need to have all sides of the k-axis grid be equal we first *square* the grid; that is we map the k-axis grid with unequal sides onto a k-axis grid with with equal sides. Alielunas and Rosenberg [2] show that two axis grids can be squared with constant dilation and expansion and Kosaraju and Atallah [18] extend this result to k-axis grids.

Combining the grid squaring with Corollary 1 gives us the following:

Corollary 2 *The $L_1 \times L_2 \cdots \times L_k$ grid can be embedded in Q_{kL} with width $\lfloor \lceil \log L \rceil /2 \rfloor$, $O(1)$ expansion, and $\lceil \lceil \log L \rceil /2 \rceil$-packet cost $O(1)$, where $L = \lceil (\prod_{i=1}^{k} L_i)^{1/k} \rceil$.*

We leave to the reader the proof of Corollary 2 and to show that the embeddings of Theorem 2 can be used to create load–2^k embeddings of k–axis grids which more fully use the edges of the hypercube.

5 Multiple–copy embeddings of Cube-Connected-Cycles

We now turn from grids to the cube-connected-cycle network (CCC). In this section we will assume that we are given a CCC and required to embed it in the smallest hypercube having at least as many nodes as the CCC. Since the m–level CCC has $m2^m$ nodes by making the number of dimensions in the hypercube be $m + \lceil \log m \rceil$ we achieve optimal expansion. In Section 6 we will be given Q_n and asked to embed the largest CCC having no more than 2^n nodes. When there exists m such that $n = m + \lceil \log m \rceil$ then the expansion will again be optimal. When no such m exists there will be an m with $n - 1 = m + \lceil \log m \rceil$. Thus though a single CCC embedding will have expansion 2 we can embed two CCCs, each in a separate copy of Q_{n-1}, so that each has optimal expansion within its subcube.

Our main result in this section, a multiple–copy embedding, is stated below. Note that since the directed cube-connected-cycle has degree 2 the edge-congestion is optimal. The dilation is also optimal since when n is odd the cube-connected-cycle has odd cycles and thus dilation 2 is necessary.

Theorem 3 *n copies of the $n2^n$-node directed cube-connected-cycles network can be embedded in $Q_{n+\lceil \log n \rceil}$ with edge-congestion 2 and when n is even dilation 1 or when n is odd dilation 2.*

Before we proceed with the proof we introduce a few definitions.

5.1 Terminology

Cube-connected-cycles networks [21]

The nodes of the n–stage directed CCC network are divided into n levels and 2^n columns. Each node has a distinct address $\langle \ell, c \rangle$, $0 \leq \ell < n$ and $0 \leq c < 2^n$. The edges are divided into two sets: the straight–edges S and cross–edges C. The sets S and C are defined as follows.

$$\begin{aligned} S &= \left\{(\langle \ell, c\rangle, \langle \ell+1 \bmod n, c\rangle) \mid 0 \le \ell < n,\ 0 \le c \le 2^n - 1\right\} \\ C &= \left\{(\langle \ell, c\rangle, \langle \ell, c \oplus 2^\ell\rangle) \mid 0 \le \ell < n,\ 0 \le c \le 2^n - 1\right\} \end{aligned}$$

The operator $\oplus$ denotes bitwise exclusive-or. For $0 \le \ell < n$, cross–edges between two nodes at level ℓ and straight–edges between nodes at levels ℓ and $(\ell+1) \bmod n$ are called *level ℓ* edges. The n nodes in column c along with the straight–edges incident to them form a length n directed cycle. The cross edges form pairs of oppositely oriented directed edges.

Definition A *window* $W \subseteq \mathsf{Z}_k$ is an *ordered* subset of the dimensions of the hypercube Q_k. For any node v in Q_k, the *signature* $\sigma_W(v)$ is the concatenation of the address bits of v in the dimensions ordered by W. For example, the signature of node 01001 over the window $W = \{1, 4, 3\}$ is 110, the bits in positions 1, 4, and 3.

Definition For any sequence a we denote the prefix of length i by $\rho_i(a)$. For any two sequences a and b, we define $\lambda(a, b)$ to equal the length of the longest common prefix.

We recall that H_k is the hamiltonian cycle formed by starting at hypercube node 0^k and crossing successively the dimensions listed in the binary reflected gray code sequence G_k.

5.2 Embeddings of the CCC Network

Greenberg, Heath and Rosenberg [11] have shown that CCC networks can be embedded efficiently within boolean hypercubes. In particular, they establish the following result.

Lemma 4 [11] *The n–level CCC with $n2^n$ nodes can be embedded in $Q_{n+\lceil \log n \rceil}$ with dilation 1 when n is even, and dilation 2 when n is odd.*

In this subsection we describe the embedding of [11] in an abstract setting which will be useful later. To embed the n–stage CCC into the hypercube Q_{n+r} (where

$r = \lceil \log n \rceil$),[§] first partition the $(n + r)$ hypercube dimensions into two windows W and $\overline{W}$ where $|W| = r$, $|\overline{W}| = n$, and $W \cap \overline{W} = \emptyset$.

The CCC vertex $\langle \ell, c \rangle$ is mapped to the hypercube node with signature $H_r(\ell)$ on window W and signature c on window $\overline{W}$. A nice feature of this mapping is that level–ℓ straight-edges are mapped to hypercube edges in dimension $G_r(\ell)$. Similarly, level–ℓ cross-edges are mapped to hypercube edges in dimension $\overline{W}(\ell)$.

Observe that the embedding is completely specified by the choice of a length–r window W, a disjoint length–n window $\overline{W}$, and a hamiltonian cycle H_r. Recall that windows are ordered sequences, so the choice of W does not completely specify $\overline{W}$.

5.3 The Multiple–copy embedding

In order to embed n copies of the CCC network into Q_{n+r}, we specify, for each copy, two disjoint windows and a hamiltonian cycle. In this subsection we show how to make these choices such that the overall edge-congestion is at most 2. Since each copy is embedded as described in the previous subsection, the dilation is 1 for each embedding.

To show that naive choices are insufficient, we consider two extremes. First, suppose that we choose the same partition of hypercube dimensions for all n copies. In all n embeddings the straight-edges are mapped to the same set of r dimensions. Consequently, the edge-congestion is at least n/r.

Next, suppose instead that each copy, i, $0 \leq i < n/r$, uses a distinct set of dimensions for its length–r window W_i. The proof that there will again be congestion n/r typifies the arguments throughout the rest of this section. We pick a dimension and, among all copies, look at all the CCC vertices which use this dimension for their cross–edge. If many of these CCC vertices are mapped onto the same hypercube node then the congestion due to cross–edges is high.

Consider one of the r dimensions, call it d, not contained in any W_i. In each copy, edges in dimension d are images of cross–edges. Furthermore, for a given copy i, all CCC vertices whose cross–edge is mapped to dimension d are at the same CCC level. It is easily established that all the hypercube images of CCC vertices on one level under embedding i have the same signature, call it s_i, on W_i.

[§] For sake of convenience, we assume in the remainder of this section that n is a power of 2. For other values of n, the congestion for multiple–copy embeddings is, at worst, doubled and some edges suffer dilation 2.

Since the windows are disjoint there exists a hypercube node, v, for which $\forall i$, $\sigma_{W_i}(v) = s_i$. Each copy maps a CCC vertex to v and the cross–edges emanating from each of these n/r CCC vertices are all mapped to dimension d. The congestion on dimension d edges is therefore n/r.

In the remainder of this section we present an n-copy embedding which avoids such congestion. In particular, we establish that every hypercube edge is the image of at most one CCC cross–edge.

Overlapping windows From the preceding discussion we know that the windows must be chosen carefully to avoid high congestion. We construct the length–r windows for the n copies as follows: all windows contain dimension 1; half of the windows contain dimension 2 and the other half contain dimension 3; in general, of all the windows that contain dimension i, half of them also contain dimension $2i$ and the other half contain dimension $2i + 1$.

Multiple embedding The length–r window and the length–n window for the kth copy are denoted $\mathcal{W}^k$ and $\overline{\mathcal{W}}^k$; the ith element of these are denoted $\mathcal{W}^k(i)$ and $\overline{\mathcal{W}}^k(i)$. Similarly, let $\mathcal{H}^k$ denote the hamiltonian cycle for the kth copy, and let $\mathcal{H}^k(i)$ be the ith node on the cycle.

We define $\mathcal{W}^k$, $\overline{\mathcal{W}}^k$, and $\mathcal{H}^k$ $(0 \leq k, \ell < n, 0 < i < \log n)$ formally as follows. (Again let $\oplus\, b(k)$ denote bitwise xor with the $\log n$–bit binary representation of k.)

$$\begin{aligned}
\mathcal{W}^k(0) &= 1 \\
\mathcal{W}^k(i) &= 2^i + \rho_i(k) \\
\overline{\mathcal{W}}^k(\ell) &= \begin{cases} \ell & \text{if } \ell \notin \mathcal{W}^k \\ n + \lfloor \log \ell \rfloor & \text{if } \ell \in \mathcal{W}^k \end{cases} \\
\mathcal{H}^k(\ell) &= H_r(\ell) \oplus b(k)
\end{aligned}$$

To show that the above defines n embeddings, we observe that $\forall k$, $\mathcal{W}^k \cap \overline{\mathcal{W}}^k = \emptyset$ and $\mathcal{H}^k$ is a Hamiltonian cycle. We leave the reader to verify these and the following observations.

Observations

The following properties of the embeddings will be useful later.

1. $\forall k, \ell$ the kth embedding maps every CCC vertex at level ℓ to a hypercube node whose signature on $\mathcal{W}^k$ equals $\mathcal{H}^k(\ell)$.

2. In the kth embedding, level–ℓ straight–edges are mapped to dimension $\mathcal{W}^k(G_r(\ell))$ hypercube edges.

3. In the kth embedding, level–ℓ cross–edges are mapped to dimension $\overline{\mathcal{W}}^k(\ell)$ hypercube edges.

We will also use the following properties of prefixes.

4. For any two embeddings k_1 and k_2: $\lambda(\mathcal{W}^{k_1}, \mathcal{W}^{k_2}) = \lambda(k_1, k_2) + 1$.

5. For any level ℓ and any two embeddings k_1 and k_2:
$\lambda(\mathcal{H}^{k_1}(\ell), \mathcal{H}^{k_2}(\ell)) = \lambda(k_1, k_2)$.

6. For any two levels ℓ_1, ℓ_2:
$\lambda(H(\ell_1), H(\ell_2)) = \lambda(\ell_1, \ell_2)$.

As mentioned earlier a frequent technique in the proof will be to identify all the CCC vertices whose cross–edge (or straight–edge) is mapped to hypercube edges in a particular dimension. We show that the congestion on hypercube edges in this dimension is small by showing that no hypercube node can be the image of more than one or two of these CCC vertices.

Observations 2 and 3 are especially useful in identifying groups of CCC vertices whose images use the same hypercube dimension for cross–edges because they associate CCC levels with the hypercube dimension to which the CCC edges are mapped. For example, observation 3 and the definition of $\overline{\mathcal{W}}^k$ together show that all cross–edges mapped to hypercube edges in dimension i, $0 \leq i < n$, are at level i. Thus we ask how many level–i CCC vertices can be mapped to a single hypercube node.

Lemma 5 *For any level i, $0 \leq i < n$, and any hypercube node v at most one of the n embeddings defined above maps a level–i CCC vertex to v.*

Proof: A dimension *separates* two sets of hypercube nodes if and only if the value of the hypercube address bit corresponding to this dimension equals 1 for all the nodes in one set and equals 0 for all the nodes in the other set. We show that for any two embeddings there exists a dimension which separates the set of hypercube images of level–i CCC vertices under one embedding from the set of images of level–i vertices under the other embedding. Thus there is no hypercube node to which two embeddings map level–i CCC vertices.

Given any two embeddings, k_1 and k_2, call the images of level–i CCC vertices under the two embeddings V_1 and V_2, respectively. Observation 1 shows that $\mathcal{H}^{k_1}(i)$ is the signature on $\mathcal{W}^{k_1}$ for all nodes in V_1 and that $\mathcal{H}^{k_2}(i)$ is the signature on $\mathcal{W}^{k_2}$ for all nodes in V_2. Observation 5 shows that $\mathcal{H}^{k_1}(i)$ and $\mathcal{H}^{k_2}(i)$ differ on their $\lambda(k_1, k_2)+1$st bit. Furthermore Observation 4 shows that the hypercube dimension in position $\lambda(k_1, k_2) + 1$ is the same for both $\mathcal{W}^{k_1}$ and $\mathcal{W}^{k_2}$.

Thus this hypercube dimension in position $\lambda(k_1, k_2) + 1$ of both length–r windows separates V_1 and V_2 as desired. ▮

Lemma 5 treated CCC vertices whose cross–edge is mapped to a hypercube dimension less than n. We next examine all CCC vertices whose cross–edge is mapped to a dimension greater than n.

Lemma 6 *For any j, $0 \leq j < r$ and hypercube node v at most one of the n embeddings maps to v a CCC vertex whose cross–edge is mapped to a dimension $n + j$ edge.*

Proof: Similarly to the previous lemma let k_1 and k_2 be any two embeddings and V_1 be the set of hypercube nodes which are images of CCC vertices using dimension $n + j$ as cross–edges under embedding k_1 and V_2 be the images of CCC vertices using dimension $n + j$ as cross–edges under embedding k_2. We will show that the hypercube dimension in position $\lambda(k_1, k_2) + 1$ of both length–r windows again separates V_1 from V_2.

Let the levels at which dimension $n + j$ is used for a cross–edge in the two embeddings be ℓ_1 and ℓ_2 respectively. When $j = 0$ and thus $\ell_1 = \ell_2 = 1$ we can apply Lemma 5.

When $j > 0$ we observe from the definition of the embeddings that $\ell_1 = 2^j + \rho_j(k_1)$ and $\ell_2 = 2^j + \rho_j(k_2)$. From these representations we see that ℓ_1 and ℓ_2 share $0^{r-j-1}1$

as their first $r-j$ bits and then $\lambda(k_1,k_2)$ more bits from the prefixes of k_1 and k_2. Thus $\lambda(\ell_1,\ell_2)=r-j+\lambda(k_1,k_2)$ and, since $r>j$, $\lambda(\ell_1,\ell_2)>\lambda(k_1,k_2)$.

Applying Observation 6 to the last equation yields $\lambda(H_r(\ell_1),H_r(\ell_2))>\lambda(k_1,k_2)$. Using the definition of $\mathcal{H}^k$ to write $\mathcal{H}^{k_1}=H_r\oplus k_1$ and $\mathcal{H}^{k_2}=H_r\oplus k_2$ we then infer that $\lambda(\mathcal{H}^{k_1}(\ell_1),\mathcal{H}^{k_2}(\ell_2))=\lambda(k_1,k_2)$. Since $\lambda(k_1,k_2)$ is the length of the longest common prefix $\mathcal{H}^{k_1}(\ell_1)$ and $\mathcal{H}^{k_2}(\ell_2)$ differ on their $\lambda(k_1,k_2)+1$st bit.

Now since by Observation 1, $\mathcal{H}^{k_1}(\ell_1)$ and $\mathcal{H}^{k_2}(\ell_2)$ are the signatures on the length–r windows of vertices in V_1 and V_2 respectively it follows that the hypercube dimension in position $\lambda(k_1,k_2)+1$ of both length–r windows separates V_1 and V_2. ∎

Putting lemma 5 and lemma 6 together we show that the congestion on cross–edges is small.

Lemma 7 *The congestion due to cross–edges is at most 1, in dimension 1 the congestion is 0.*

Proof: The congestion in dimension 1 due to cross–edges is 0 since no cross–edges are mapped to dimension 1 edges. For dimension 0 or dimension d, $2\le d<n$, Lemma 5 guarantees that at most one CCC vertex using dimension d is mapped to any hypercube node. Similarly Lemma 6 guarantees that for $d>n$, at most one CCC vertex using dimension d is mapped to any hypercube node. The single CCC vertex mapped to a hypercube node using a given dimension for its cross–edge contributes congestion of one. ∎

We next turn our attention to straight–edges. From observation 2 we know that embedding k maps level–ℓ straight–edges to hypercube dimension $\mathcal{W}^k(G_r(\ell))$. From the definition of G_r we can invert this relation to determine which levels are mapped to a given hypercube dimension. For example, dimension 1 is always the first element in $\mathcal{W}^k$ and thus always corresponds to the most significant bit in the gray code. Since the most significant bit is used only at $G_r(n/2-1)$ and $G_r(n-1)$ we can conclude that only straight–edges at level $n/2-1$ and $n-1$ are mapped to dimension 1 hypercube edges.

The remaining dimensions can be divided into r tiers: dimension i is in tier $t=\lfloor\log i\rfloor$. A tier t dimension will always correspond to bit t in the gray code (with msb $=0$). A simple fact about reflected gray codes is that bit $t>0$ is used 2^t times and that for any two levels at which t is used there exists some bit $t'<t$ which is used an odd number of times between these two levels.

Lemma 8 *For any dimension i, $1 < i < n$, and hypercube node v at most one of the n embeddings maps to v a CCC vertex whose straight–edge is mapped to a dimension–i hypercube edge. Furthermore no more than two embeddings map to v a CCC vertex whose straight–edge is mapped to a dimension 1 edge.*

Proof: Given two embeddings k_1 and k_2, define V_1 to be the images of all CCC vertices which use dimension i for straight–edges under embedding k_1 and V_2 be the images under embedding k_2. If i is in tier t and both V_1 and V_2 are nonempty then $\lambda(\mathcal{W}^{k_1}, \mathcal{W}^{k_2}) > t$. (From the definition of $\mathcal{W}^k$ if two embeddings have the same dimension in position t they must have the same dimension in all positions j, $0 \leq j \leq t$.)

Since i is in tier t the nodes in V_1 and V_2 can only be the images of CCC vertices in the 2^t levels at which tier–t dimensions are used for straight edges. Partition the nodes of V_1 and V_2 into subsets depending on the CCC level of their preimages. Thus V_{1,ℓ_1} is the subset of V_1 containing images of level–ℓ_1 CCC vertices and V_{2,ℓ_2} is the subset of V_2 containing images of level–ℓ_2 CCC vertices. By separating each subset of V_1 from every subset of V_2 we will show that V_1 is disjoint from V_2.

Now let ℓ_1 and ℓ_2 be two levels at which tier–t dimensions are used for straight–edges. If $\ell_1 = \ell_2$ then by Lemma 5 V_{1,ℓ_1} is separated from V_{2,ℓ_2}. On the other hand if $\ell_1 \neq \ell_2$ then because some bit $t' < t$ is used an odd number of times between the two levels it follows that $\rho_t(H_r(\ell_1)) \neq \rho_t(H_r(\ell_2))$. Since $\lambda(k_1, k_2) > t$ we can use the definition of $\mathcal{H}^k$ to infer from the previous equation that $\rho_t(\mathcal{H}^{k_1}(\ell_1)) \neq \rho_t(\mathcal{H}^{k_2}(\ell_2))$. Let $j \leq t$ be one position where the two prefixes differ.

As in the previous lemmas we use Observation 1 to show that the signatures on the length–r windows of nodes in V_{1,ℓ_1} and V_{2,ℓ_2} are $\mathcal{H}^{k_1}(\ell_1))$ and $\mathcal{H}^{k_2}(\ell_2))$. Then since $\lambda(\mathcal{W}^{k_1}, \mathcal{W}^{k_2}) > t$ the dimension in position j of both windows separates V_{1,ℓ_1} from V_{2,ℓ_2}.

It is also easily verified that dimension 1 is used for straight–edges at level $n/2 - 1$ and $n - 1$ in each embedding. Thus by Lemma 5 at most one embedding for each of these two levels can map to v a CCC vertex whose straight–edge is mapped to a dimension 1 hypercube edge. ▮

To complete the proof of Theorem 3 we note that edges in dimension 1 are never used for cross–edges and are used at most twice for straight–edges while edges in other dimensions are used at most once for straight–edges and once for cross–edges. Thus the overall congestion of two is established.▮

5.4 Extensions

If an undirected version of the CCC is desired the straight–edges directed toward the lower level must also be included. By a variant of Lemma 8 these edges will contribute an additional congestion of at most two increasing the total congestion to four.

A corollary of Theorem 3 is that every graph which is efficiently embeddable within a CCC network has efficient multiple–copy embeddings within the hypercube. It is easy to show that FFTs and Butterflies can be embedded in CCCs with dilation 2 and congestion 2. Thus they also have efficient multiple–copy embeddings into the hypercube.

6 A General Technique

In this section we extend the techniques of Section 4 to a more general setting. In particular, starting with an n–copy embedding of a directed graph G into Q_n the general technique produces a width–n embedding of 2^{n+1} copies of G into Q_{2n}. This transformation has the property that if the cost of the multiple–copy embedding is c, and δ is the maximum outdegree of any vertex of G, then the n–packet cost of the multiple–path embedding is $c + 2\delta$.

For example, in Section 4 we started with a multiple–copy embedding of cycles and produced an embedding of 2^{n+1} width–n cycles. The cost of the multiple–copy embedding is 1, the outdegree of each vertex in a directed cycle is 1, and consequently the n–packet cost of the multiple–path embedding is 3.

At the end of this section we will apply the general technique to the multiple–copy embedding of the butterfly network. The resulting width–n graph has the property that it yields a width–n embedding of the complete binary tree. In Section 7 we discuss how the same width–n graph can also be used for fast bit-serial routing in a manner similar to the 'dilated' butterflies of Aiello, Leighton, Maggs and Newman [1].

We start with a generalization of the cross-product of two graphs to the cross-product of two sets of graphs. Let $\mathcal{R} = \{R_i \mid i \in Z_N\}$ and $\mathcal{C} = \{C_i \mid i \in Z_N\}$ be two sets of graphs, such that each R_i, C_i has vertex set Z_N.

The *cross product of two sets* $\mathcal{R}$ and $\mathcal{C}$ is the graph (V, E) where $V = Z_N \times Z_N$, and the edge set E is defined to be

$$E = \{(\langle i, j_1\rangle, \langle i, j_2\rangle) \mid i \in \mathsf{Z}_N, (j_1, j_2) \in R_i\} \cup \{(\langle i_1, j\rangle, \langle i_2, j\rangle) \mid (i_1, i_2) \in C_j, j \in \mathsf{Z}_N\}$$

One can visualize the vertices as arranged in an $N \times N$ grid. The edges in E connect rows and columns so that the subgraph induced by row i equals R_i and the subgraph induced by column j equals C_j.

We pause for one remark regarding our terminology. We say that graph $A = (\mathsf{Z}_N, E)$ *equals* graph $B = (\mathsf{Z}_N, F)$ if and only if $E = F$. That is, the graphs are equal if and only if they are isomorphic under the identity mapping on vertices. In general, isomorphic graphs need not be equal.

It is easy to see that if, for all i, R_i equals G and C_i equals H then the generalized cross product equals the standard cross product $G \times H$. Note that this is not necessarily true when "equals" is replaced by "is isomorphic to."

In our use of the generalized cross product, the sets $\mathcal{R}$ and $\mathcal{C}$ will each contain isomorphic copies of the same graph. Before we proceed, we need one more definition. Let $G = (\mathsf{Z}_N, E)$ be a graph and $\phi : \mathsf{Z}_N \mapsto \mathsf{Z}_N$ be an automorphism on Z_N. Then the graph G_ϕ is defined as the graph with vertex set Z_N and edge set $\{(\phi(u), \phi(v) \mid (u, v) \in E\}$.

Now, consider an n–copy embedding of some graph $G = (\mathsf{Z}_N, E)$ in Q_n. Number these copies 0 through $n - 1$. Each copy is an isomorphic image of G. In other words, the ith copy defines an automorphism ϕ_i of Z_N, such that $\phi_i(j)$ is the address of vertex j in the hypercube under the ith copy.

Having fixed ϕ_i, $0 \le i < n$, let $R_i = C_i = G_{\phi_{M(i)}}$, where $M(i)$ is, of course, the moment of the number i. Finally, define the *induced cross product* $X(G)$ to be the generalized cross product of the sets $\mathcal{R}$ and $\mathcal{C}$.

Theorem 4 *Let G be a graph with maximum out degree δ, and for which there is an n–copy embedding in Q_n with cost c. Then there exists a width–n embedding of $X(G)$ into Q_{2n} with n–packet cost $c + 2\delta$.*

Proof: The vertex embedding follows directly from the definition of $X(G)$. First we divide Q_{2n} into the cross product $Q_n \times Q_n$. Next, as in the proofs of Theorem 1 and 2 we view the cross product as a grid with a copy of Q_n on each row and column. The rows are named by the most significant n bits of their hypercube addresses and the columns by the least significant n bits. Finally we embed R_i in row i and C_j in column j via the identity mapping.

Each edge of $X(G)$ embedded in a row is given width n by replacing it with the following n length–three paths. Suppose the edge is mapped to hypercube edge (x, y) in dimension d, $0 \leq d < n$. For $0 \leq k < n$ the kth path for (x, y) is $x, x \oplus 2^{n+k}, x \oplus 2^{n+k} \oplus 2^d, y$. Intuitively it crosses into a neighboring row, follows the projection of (x, y) in this neighboring row's subcube, and then crosses back into the original row. Similarly if an edge of $X(G)$ is embedded in a column it is mapped to some hypercube edge (u, v) in dimension $n + d$, $0 \leq d < n$. The kth path for (u, v) is $u, u \oplus 2^k, x \oplus 2^k \oplus 2^{n+d}, y$.

It remains to bound the cost of the embedding. We first note that together the first edges of all the paths use each directed hypercube edge at most δ times. (Each hypercube node has at most δ edges from R_i in its row special image and uses each directed edge in the column dimensions once for each such edge. The edges for the C_j place the same load on the row dimensions.) Similarly the final edges of all the paths also use each directed hypercube edge at most δ times.

To complete the proof we must show that the cost of the middle edges is bounded by the cost of the n–copy embedding of G. Consider all the middle edges in a particular row. They are each the projection of an edge of $X(G)$ from a neighboring row. Lemma 2 and the construction of $X(G)$ guarantee that each neighboring row is a different automorph of G. Furthermore the isomorphs of G were constructed so that their combined projections formed the n–copy embedding of G in Q_n. Thus together all the middle edges in this row form the n–copy embedding and can be simulated with cost c. Similarly the middle edges in each column also form an n–copy embedding.

Thus a simulation of the entire multiple–path embedding takes δ steps to simulate all first edges, c steps to simulate all middle edges, and δ steps to simulate all final edges. ∎

6.1 Complete Binary Trees

Theorem 5 *For all m and $n = m2^m$ the $(2^{2n} - 1)$–vertex complete binary tree can be embedded in Q_{2n} with width n, $O(1)$ n–packet cost, and $O(1)$ load.*

Proof: Section 5 shows that m copies of the butterfly can be embedded in Q_n with $O(1)$ cost. By repeating $n - m$ copies twice the n–copy embedding with $O(1)$ cost required by Theorem 4 is achieved. When Theorem 4 is applied using this multiple–copy embedding of the butterfly the result is a width-n embedding of a generalized cross product of butterflies (call the product $\mathcal{X}$) in Q_{2n} with $O(1)$ n–packet cost.

We next show that the $2n$–level complete binary tree (CBT) can be embedded in $\mathcal{X}$ with $O(1)$ congestion, dilation, and load. Our main tool will be the fact that the M–node CBT can be embedded in the M–node butterfly with $O(1)$ congestion, dilation, and load [4]. The embedding is simplified by the fact that the embedding in [4] never maps two CBT leaves to the same butterfly node.

We start by embedding the top n levels of the CBT into R_0, the butterfly along the top row of $\mathcal{X}$. Each column of $\mathcal{X}$ receives at most one level–n CBT vertex. The tree is then extended by treating each level–n vertex as the root of a n level CBT. These subtrees are each embedded in the butterfly corresponding to the column of the root. Since the leaf level of the row tree and the root level of the column trees is the same we have an embedding of a $(2n-1)$–level complete binary tree. We complete the embedding by giving each leaf of a column tree two children in its row's butterfly. One child is mapped to the butterfly neighbor along the cross–edge and the other along the straight–edge to the next higher butterfly level.

The mapping of the first n CBT levels has load equal to that of the embedding in [4] since they correspond to a single CBT embedding. The next $n-1$ levels have the same load since each subtree is embedded in its own column butterfly. In addition each hypercube may have two CBT leaf nodes mapped to it so the overall load is 2 plus the load due to the embedding of [4]. The edges of $\mathcal{X}$ are each used at most once by the first $2n-1$ CBT levels and once by the last level. Thus the overall congestion on the edges of $\mathcal{X}$ is at most twice the congestion of the CBT to butterfly embedding of [4]. In sum, the n–packet cost and the load of the CBT to hypercube embedding are both $O(1)$. ▮

When a multiple–path embedding of the n–level CBT, where n is not of the form specified by the theorem, is desired a more complicated construction is necessary. For these cases the butterflies will not map bijectively to the factor hypercubes. Thus the embeddings will have larger expansion. In addition the simple approach of embedding first the top n levels and then the bottom n levels of the CBT may not work since the level n nodes may not be mapped to the images of butterfly nodes. While O(1) load and cost is still achievable we omit the proof from this paper.

6.2 Arbitrary binary trees

In [6] it is shown that any $(2^n - 1)$–node, constant degree tree can be embedded in an n–level complete binary tree with $O(\log n)$ congestion and dilation. By composing this embedding with the multiple–path CBT embedding we achieve a width–n embedding of arbitrary constant degree trees into hypercubes with cost $O(\log n)$. All our previous embeddings had given us $O(n)$ speed-ups over standard embeddings while this embeddings yields only $O(n/\log n)$ speed-up.

7 Applications to bit-serial message routing

In the previous section we used the multiple–path embedding of the induced cross-product of butterflies, $\mathcal{X}$, to produce multiple–path embeddings of trees. In this section we examine applications to message routing in hypercubes.

There are a number of fast randomized algorithms for permutation routing on the FFT, CCC and butterfly networks [17, 20, 23]. These are all "store-and-forward" algorithms, in which each message packet can be forwarded over a link in unit time. On the $n2^n$–node butterfly network for example, each of the nodes can send a packet to a unique destination so that, with overwhelmingly high probability, each packet is delivered in $O(n)$ time steps. This implies that, with high probability, each packet spends $O(n)$ time steps waiting in queues.

With longer messages and bit-serial routing, "cut-through" or "wormhole" routing [9, 10] is typically used to forward messages. Rather than being queued at a single node a message can span several nodes; the entire message is piped along the same path from source to destination but depending on queuing delays the distance between the front and rear of the message can grow and shrink. Under this model and with messages containing M packets, the store-and-forward algorithms mentioned above may require a message to wait M steps each time it is queued. Thus if each message is M packets in length, then, with high probability, the algorithms mentioned above will complete delivery of all messages in $\Theta(nM)$ time steps.

The multiple–copy embedding of the CCC network allows wormhole routing to be speeded on the hypercube. By breaking up each message into n distinct pieces, and routing n permutations on the n copies of the CCC, the time to completion is reduced to $O(M)$.

In a recent paper, Aiello, Leighton, Maggs and Newman [1] consider two different switch models for bit–serial routing on the hypercube. In the strong switch model, each switch can permute n incoming messages onto n outgoing channels in one time step. In the weak model only $O(1)$ incoming messages can be permuted onto outgoing channels in one time step although once the connection is established it can be continued on later steps without counting as a new permutation. They show that, under both models, M–bit message permutations can be completed bit-serially in $O(M)$ steps with high probability. This is accomplished via an embedding of the *dilated butterfly* onto the hypercube [1]. In what follows we give a simpler embedding of the same network.

The dilated butterfly is a width n, $O(1)$ congestion embedding of the butterfly in which all edges (except those at two levels of the butterfly) have $O(1)$ dilation. Edges in the two special levels can have dilation up to $2n$.

The multiple–path embedding of $\mathcal{X}$ gives a simple multiple–path embedding of the butterfly. Butterfly edges between levels $n/2$ and $n/2+1$ and between levels $n-1$ and 0 are cut, thereby decomposing the butterfly into two sets of $2^{n/2}$ independent $n/2$–level butterflies. One set is mapped to the rows and the other to the columns of $\mathcal{X}$. The cut edges are inserted next; while these have width n, they can have dilation up to $2n$. The pairing of endpoints of cut edges forms a partial permutation. We need to find n disjoint routes for each cut edge in the partial permutation. The partial permutation can be routed along disjoint paths of length $2n$ in each of the n copies of an n–copy embedding of the CCC in the entire hypercube. This gives the desired embedding.

A better alternative is to use the width-n embedding of $\mathcal{X}$ directly to route messages. Each route takes two phases; in the first phase each message is routed along a row butterfly into the column butterfly of the destination. In the second phase the message is routed along the column butterfly to reach the destination. With a fast randomized routing algorithm, each route suffers delay $O(n)$. By using the multiple–paths corresponding to each width–n edge of the $\mathcal{X}$, the need to queue messages can be eliminated and wormhole or cut-through routing in the strong-switch model is possible.

8 Summary and Comparison of Embeddings

8.1 Multiple–path, Multiple–copy, and Large–copy Embeddings

Multiple–path embeddings We have presented multiple–path embeddings for several common communication graphs: cycles, grids, trees, butterflies, FFTs and CCCs. The embedding for cycles had optimal dilation and cost; grids and complete binary trees had $O(1)$ cost while arbitrary N–node trees had cost $O(\log \log N)$. The CCC, FFT, and Butterfly embeddings had some high dilation edges but since they were confined to two levels they still allowed wormhole routing algorithms to be simulated.

Multiple–copy embeddings We have presented a multiple–copy embedding of CCCs and noted the existence of multiple–copy embeddings of cycles. Multiple–copy embeddings of grids can be formed from the multiple–copy embeddings of cycles by the same *squaring* technique[2, 18] combined with cross product decomposition used to convert the multiple–path embeddings of cycles to multiple–path embeddings of grids. Multiple–copy embeddings of trees are obtained by applying the embeddings of trees into CCC [5, 4] to the multiple–copy embeddings of the CCC.

Large–copy embeddings If many guest graph nodes are mapped to a single host node, as with multiple–copy embeddings, there is an alternative method of using all hypercube edges. A single large copy, (containing $n2^n$ nodes), can be embedded into Q_n so that the $n2^n$ vertices are evenly balanced over the 2^n hypercube nodes and the $O(n2^n)$ guest edges are evenly divided among the $n2^n$ directed hypercube edges. We will call such an embedding a *large-copy* embedding. Johnsson and Ho have used large–copy embeddings of grids to speed matrix operations[15, 16].

Large–copy embeddings of cycles are easy to construct. A large cycle is embedded by traversing the edge-disjoint cycles of Lemma 1 in sequence. Each traversal of an edge-disjoint cycle in the hypercube embeds 2^n vertices of the large cycle and no two traversals use the same edge. Thus we get the following corollary:

Corollary 3 *For even n the $n2^{n-1}$-node undirected cycle and the $n2^n$-node directed cycle can each be embedded in Q_n with dilation 1 and congestion 1.*

Large–copy embeddings of CCCs, FFTs, and butterflies are also simple to derive. Standard constructions of each of these graphs expand each node of Q_n into an n–node cycle or path. The n edges associated with each hypercube node are divided among the n nodes which replace it. Thus the degree is reduced to a constant (three for the CCC and four for FFTs and butterflies). The new graph has n times as many nodes as the original hypercube.

When embedding the $n2^n$ node FFT-like graph into Q_n the construction above is simply reversed. Each n–node cycle or path is mapped to the hypercube node from which it was expanded. The edges of the FFT-like graph are spread out evenly among the hypercube edges and an efficient large–copy embedding results.

Lemma 9 *The $n2^n$–node CCC, FFT, and butterfly can be embedded in Q_n with dilation 1 and congestion 1 for the CCC and congestion 2 for the FFT and butterfly.*

The embeddings in [4] and [6] can again be applied to yield large–copy embeddings of trees from the large copy embeddings of FFTs.

8.2 Comparisons

Many problems can be solved via several algorithms, each with a different communication graph. Thus, though multiple–path, multiple–copy, and large–copy embeddings apply to, respectively, a single 2^n–node graph, n copies of a 2^n–node graph, and a single $n2^n$–node graph, it is often necessary to compare the efficacy of the different embeddings.

Multiple–copy embeddings of cycles and the large–copy embeddings of cycles and of CCCs have unit dilation and congestion. Thus the communication speed-up is dependent only on the ability of each hypercube node to process messages on all its links at once. No forwarding of messages is necessary. On the other hand each host node must time–slice the computation for n guest nodes.

In comparison, the multiple–path embeddings require little or no time–slicing of computation but the paths are of length at least three. Thus the hypercube node must be able to forward messages. The distance three paths also result in the cost of the multiple–path embeddings being larger than the cost of the large–copy or multiple–copy embeddings.

In general the relative benefits of one method against another depend on the relative speed of communication versus computation, on the speed and sophistication of the routing circuitry within a node, and on the efficiency with which algorithms can be designed to use each method. We conclude this section with one example of the algorithmic issues in choosing an embedding.

8.3 Mapping Large Grids

We return to the example of a large grid relaxation from Section 2. We assume that there are many more grid points than there are available hypercube processors and that the relative speed of computation and communication makes it efficient to use all the hypercube processors. We must determine how to map the grid points to the processors.

Suppose the grid has M^2 points and there are N^2 processors. A first approach is to treat each grid point as a process and use the large–copy embedding. Each hypercube processor is the image of M^2/N^2 grid points, each hypercube link is the image of $O(M^2/(N^2 \log N))$ paths, and each process sends the data for one point per communication phase.

A second approach is to divide the grid up into N^2 blocks by grouping the points into $M/N \times M/N$ squares. Now each process corresponds to the updating of a $M/N \times M/N$ region of grid points and the communication graph for the processes is a $N \times N$ grid. If a multiple–path embedding is used then each hypercube processor is the image of one process, each link is the image of $O(1)$ paths, and each process sends, during one communication phase, data for the $O(M/N)$ grid points along the sides of its region.

A third approach is to divide the grid up into a $N \log N \times N \log N$ grid of $M/(N \log N) \times M/(N \log N)$ squares. The large–copy embedding can then be used; resulting in $\log^2 N$ processes per hypercube processor, $\log N$ paths per hypercube link, and data from $M/(N \log N)$ points sent per communication phase.

Computationally all three approaches are equivalent; each must compute the new value of M^2/N^2 grid points at each hypercube processor for each phase. The amount of communication necessary is not, however, equal. The first approach requires the value of $O(M^2)$ grid points to be communicated to neighboring hypercube nodes. The latter two approaches reduce the communications traffic by mapping neighboring grid points to the same processor, thus they require the communication of, respectively, $O(MN)$ and $O(MN \log N)$ grid point values.

Thus the multiple–path embedding allows the total communication to be minimized. However, for small values of N the increase of total communication by a factor of $\log N$ for the third approach as compared to the second may be smaller than the decreased cost of using a large–copy embedding instead of a multiple–path embedding. Asymptotically the multiple–path approach is best but in practice the blocked large–copy approach may be competitive.

9 Open Questions

We have shown how to use the communication resources of hypercubes more efficiently. In particular, if a computation is communication limited and employs a grid, binary tree, or FFT-like graph as its communication pattern then a savings of a factor of $O(n)$ can be realized. The multiple communication paths can also be used to increase tolerance of communication faults and to allow fast bit-serial routing.

Although the multiple–path embedding of the cycle has optimal efficiency the embeddings of multi-dimension grid and arbitrary tree embeddings could be improved. For the grids two open questions are unresolved. What is the best way to implement grids whose sides are (1) unequal, or (2) not equal to powers of 2? Are there embeddings which use all links even when communication proceeds along one grid axis at a time? Similarly, for arbitrary binary trees it remains unknown whether the speed–up is necessarily a factor $\log n$ worse than for complete binary trees.

Acknowledgements

Charles Leiserson helped simplify an earlier, and weaker, version of Theorem 1. We thank him for his suggestions. We thank Arny Rosenberg and Lennart Johnsson for helpful discussions and for pointing out fruitful avenues. We also thank Ajit Agrawal and Ching-Tien Ho for illuminating discussions. The authors were supported by the NSF/DARPA grant CCR-8908285, NSF grant CCR-8807426, and AFOSR grant 89-0382.

References

[1] W. Aiello, F.T. Leighton, B. Maggs, and M. Newman. Fast algorithms for bit–serial routing on a hypercube. MIT typescript 1990.

[2] R. Aleliunas and A. L. Rosenberg. On embedding rectangular grids in square grids. *IEEE Trans. Comp.*, 31:907–913, 1980.

[3] B. Alspach, J-C. Bermond, and D. Sotteau. Decomposition into cycles i: Hamilton decompositions. Technical Report 87-12, Simon Fraser University, 1987.

[4] S.N. Bhatt, F.R.K. Chung, J.-W. Hong, F.T. Leighton, and A.L. Rosenberg. Optimal simulations by butterfly networks. In *20th Annual ACM Symposium on Theory of Computing*, pages 192–204, 1988.

[5] S.N. Bhatt, F.R.K. Chung, F.T. Leighton, and A.L. Rosenberg. Optimal simulations of tree machines. In *27th Annual Symposium on Foundations of Computer Science*, pages 274–282, 1986.

[6] S.N. Bhatt, F.R.K. Chung, F.T. Leighton, and A.L. Rosenberg. Universal graphs for bounded–degree trees and planar graphs. *SIAM J. on Discrete Math*, 2.2:145–155, 1989.

[7] M.Y. Chan. Dilation–2 embeddings of grids into hypercubes. In *Int. Conf. on Parallel Processing*, pages 295–298, 1988.

[8] M.Y. Chan. Embedding of d–dimensional grids into optimal hypercubes. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 52–57, 1989.

[9] W. J. Dally. A VLSI architecture for concurrent data structures. Technical Report 5209, California Institute of Technology, 1986.

[10] W.J. Dally and C.L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. Comp.*, 36:547–553, 1987.

[11] D. S. Greenberg, L. S. Heath, and A. L. Rosenberg. Optimal embeddings of butterfly-like graphs in the hypercube. *Math. Syst. Th.*, to appear, 1990.

[12] I. Havel and P. Liebl. Embedding the polytomic tree into the n-cube. *Časopis pro Pěstován ́ı matematiky*, 98:307–314, 1973.

[13] D. Hillis. *The Connection Machine.* MIT Press, 1985.

[14] C.-T. Ho and S.L. Johnsson. Spanning balanced trees in boolean cubes. *SIAM J. Sci. Statist. Comput.*, to appear, 1990. also as Yale University Technical Report 508, 1987.

[15] S. L. Johnsson. Communication efficient basic linear algebra computations on hypercube architectures. *J.Par. and Dist. Comp.*, 4:133–172, 1987.

[16] S. L. Johnsson and C.-T. Ho. Multiplication of arbitrarily shaped matrices on boolean cubes using the full comunications bandwidth. Technical Report 721, Yale University, July 1989.

[17] A. Karlin and E. Upfal. Parallel hashing — an efficient implementation of shared memory. In *18th Annual ACM Symposium on Theory of Computing*, 1986.

[18] S. R. Kosaraju and M. J. Atallah. Optimal simulations between mesh–connected arrays of processors. Technical Report 561, Purdue University, September 1986.

[19] F. T. Leighton, B. Maggs, and S. Rao. Universal packet routing algorithms. In *29th Annual Symposium on Foundations of Computer Science*, pages 256–269, 1988.

[20] N. Pippenger. Parallel communication with limited buffers. In *25th Annual Symposium on Foundations of Computer Science*, pages 127–136, 1984.

[21] F.P. Preparata and J. Vuillemin. The cube-connected cycles: A versatile network for parallel computation. *CACM*, 24.5:300–309, 1981.

[22] M. O. Rabin. Efficient dispersal of information for security, load balancing and fault tolerance. Technical Report 02–87, Harvard University, 1987.

[23] A. G. Ranade. How to emulate shared memory. In *28th Annual Symposium on Foundations of Computer Science*, pages 185–194, 1987.

[24] E. M. Reingold, J. Nievergelt, and N. Deo. *Combinatorial Algorithms: Theory and Practice*. Prentice-Hall, Englewood Cliffs, NJ, 1977.

[25] C.L. Seitz. The cosmic cube. *CACM*, 28.1:22–33, 1985.

[26] Q.F. Stout and B. Wagar. Intensive hypercube communication, I: Prearranged communication in link-bound machines. Technical Report 9-87, University of Michigan, 1987.