OPTIMAL SIMULATIONS BY BUTTERFLY NETWORKS[1]

Sandeep N. Bhatt†, Fan R.K. Chung§,
Jia-Wei Hong‡, F. Thomson Leighton¶,
Arnold L. Rosenberg
Computer and Information Science Department
University of Massachusetts

COINS Technical Report 88-55

†Yale University, New Haven CT
§Bell Communications Research, Morristown, NJ
‡Beijing Computer Institute, Beijing, CHINA
¶MIT, Cambridge, MA

[1] A preliminary version of this paper was presented at the *20th ACM Symposium on Theory of Computing*, Chicago, IL, May 2-4, 1988

OPTIMAL EMBEDDINGS OF BUTTERFLY-LIKE GRAPHS IN THE HYPERCUBE
(Extended Abstract)

David S. Greenberg
Computer Science
Yale Univ.
New Haven, CT 06520

Lenwood S. Heath
Computer Science
Virginia Tech
Blacksburg, VA 24061

Arnold L. Rosenberg
Computer and Information Science
Univ. Massachusetts
Amherst, MA 01003

Abstract

We present optimal embeddings of three genres of butterfly-like graphs in the Hypercube; each embedding is specified via a linear-time algorithm. Our first embedding finds an instance of the order-n FFT graph as a subgraph of the smallest Hypercube that is big enough to hold it, i.e., the $(n+\lceil\log_2(n+1)\rceil)$-dimensional Hypercube. This embedding yields an on-line mapping of the pipelined FFT algorithm on the Hypercube architecture, which is optimal in all resources. Our other two embeddings map each of the order-n Butterfly graph and the order-n Cube-Connected Cycles graph into the smallest Hypercube that is big enough to hold it, i.e., the $(n+\lceil\log_2 n\rceil)$-dimensional Hypercube. These embeddings, too, are optimal in all resources: They have dilation $1+(n \bmod 2)$, which is best possible.

1. INTRODUCTION

1.1. The Main Results and Motivation

We prove that the Hypercube can simulate butterfly-like communication patterns with essentially no time loss:

1. The FFT graph is a subgraph of the smallest Hypercube that is big enough to hold it.
2. Each of the Butterfly graph and the Cube-Connected-Cycles (CCC) graph is efficiently embeddable in the smallest Hypercube that is big enough to hold it. An even-order Butterfly or CCC is embeddable as a subgraph; an odd-order graph is embeddable with unit congestion, but only with dilation 2. The increased dilation is inevitable.

All three of our embeddings are specified by means of linear-time algorithms.

Motivating our first result is the problem of mapping a parallel algorithm onto a processor array, accommodating the algorithm's intertask dependence structure to the array's interprocessor communication structure. Traditionally, one views both structures as simple undirected graphs and views the mapping problem as one of finding an efficient embedding of the algorithm-graph in the array-graph [1, 2, 5]. Our first result studies the mapping problem for the *Fast*

Fourier Transform (*FFT*) algorithm, which is paradigmatic for convolution-based algorithms, in the Hypercube architecture [2, 4, 6-8, 11]. We present a family of embeddings, each of which finds an instance of the n-level FFT graph $F(n)$ in the d_n-dimensional[1] Hypercube $Q(d_n)$, as a subgraph. We also construct optimal embeddings that are *modular*, in that the embedding of $F(n+1)$ in $Q(d_{n+1})$ is an extension of the embedding of $F(n)$ in $Q(d_n)$. Our embeddings map the *pipelined FFT algorithm* onto the Hypercube architecture, *utilizing all resources optimally*; our modular embeddings are *on-line*, in that inputs can be added to the Transform at any time, even after computation has begun. Our mappings supplement earlier work which has shown the Hypercube to be an efficient host for divide-and-conquer [2] and grid-based algorithms [8], as well as for a number of specific algorithms [6, 7].

Motivating our second result is the question of how efficiently one interconnection network can simulate another. This problem, too, is frequently studied via graph embeddings: The guest graph represents the interconnection network to be simulated, and the host graph represents the simulating network [3, 4]. Our second result studies the *Butterfly* and *CCC* networks — bounded-degree approximations to the Hypercube which equal its speed on the important class of *ascend-descend* algorithms [9] Our embeddings show that the Butterfly and CCC do not possess any communication power that is not already present in the Hypercube.

A result superficially similar to our first result appears in [7], where it is shown that each single level of the FFT graph is a spanning subgraph of the Hypercube. Our result is materially harder, in that we embed the entire FFT graph into the Hypercube at once. In a somewhat similar vein, it is shown in [6] that if one stores the data for any one level of the FFT algorithm in Hypercube processors according to a binary reflected Gray code, then any two data items that are combined via a butterfly at that level of the algorithm reside at processors that are at worst at distance 2 from one another. This result, too, is much weaker than ours, because of the "distance 2" assertion as well as the fact that only one level of the algorithm is embedded at a time.

1.2. The Formal Framework

Let G and H be simple undirected graphs, having $|G|$ and $|H|$ vertices, respectively. An *embedding* of G in H is a one-to-one association of the vertices of G with the vertices of H, together with a routing of each edge of G within H. The *dilation* of the embedding is the maximum length of the routing of any edge of G; the *congestion* is the maximum number of edges of G that are routed over a single edge of H; the *expansion* is the ratio $|H|/|G|$. Clearly, a unit-dilation embedding finds an instance of G as a subgraph of H.

We focus here on embedding three finite families of graphs $\mathcal{G}_1$, $\mathcal{G}_2$, and $\mathcal{G}_3$ in a fourth finite family $\mathcal{H}$. We seek dilation- and congestion-optimal embeddings of each $G \in \mathcal{G}_i$ in the *smallest* $H \in \mathcal{H}$ that will hold it, i.e., for which $|H|/|G| \geq 1$. Thus, we optimize expansion and then try to optimize dilation and congestion. We succeed in optimizing all three cost measures

[1] $d_n =_{\mathrm{def}} n + \lceil \log(n+1) \rceil$; all logarithms are to the base 2.

simultaneously. The graph families $\mathcal{G}_i$ are FFT graphs, Butterfly graphs, and CCC graphs; the graph family $\mathcal{H}$ is the Hypercubes. Let n be a positive integer:

- The *order-n FFT graph* $F(n)$ has vertex-set[2] $V_n = Z_{n+1} \times Z_2^n$. For each vertex $v = \langle \ell, \vec{\delta} \rangle \in V_n$, we call ℓ the *level* of v and $\vec{\delta}$ the *position-within-level (PWL) string* of v. Vertices at level 0 are called *inputs*, and vertices at level n are called *outputs*. The edges of $F(n)$ are of two types: For each $\ell \in Z_n$ and $\delta_0\delta_1 \cdots \delta_{n-1} \in Z_2^n$, vertex

  $\langle \ell,\ \delta_0\delta_1 \cdots \delta_{n-1} \rangle$, on level ℓ of $F(n)$,

  is connected by a *straight-edge* with vertex

  $\langle \ell + 1,\ \delta_0\delta_1 \cdots \delta_{n-1} \rangle$, on level $\ell + 1$ of $F(n)$

  and is connected by a *cross-edge* with vertex

  $\langle \ell + 1,\ \delta_0\delta_1 \cdots \delta_{\ell-1}(1 - \delta_\ell)\delta_{\ell+1} \cdots \delta_{n-1} \rangle$, on level $\ell + 1$ of $F(n)$;

  One can view $F(n)$ inductively: $F(1) = K_{2,2}$, the *butterfly* (or, complete bipartite graph on two inputs and two outputs); for $n \geq 2$, one obtains $F(n)$ by taking two copies of $F(n-1)$ and 2^n new output vertices, and constructing butterflies connecting the k^{th} outputs of each copy of $F(n-1)$ to the k^{th} and $(k + 2^{n-1})^{\text{th}}$ new outputs.

- The *order-n Butterfly graph* $B(n)$ is $F(n)$, with "wraparound" obtained by identifying each input vertex $\langle 0,\ \vec{\delta} \rangle$ with the corresponding output vertex $\langle n,\ \vec{\delta} \rangle$.

- The *order-n Cube-Connected Cycles (CCC) graph* $C(n)$ has vertex-set $W_n = Z_n \times Z_2^n$. The edges of $C(n)$ are of two types: For each $\ell \in Z_n$ and $\delta_0\delta_1 \cdots \delta_{n-1} \in Z_2^n$, vertex

  $\langle \ell,\ \delta_0\delta_1 \cdots \delta_{n-1} \rangle$, on level ℓ of $C(n)$,

  is connected by a *straight-edge* with vertex

  $\langle \ell',\ \delta_0\delta_1 \cdots \delta_{n-1} \rangle$, on level $\ell' = \ell + 1 \pmod{n}$ of $C(n)$,

  and is connected by a *level-edge* with vertex

  $\langle \ell,\ \delta_0\delta_1 \cdots \delta_{\ell-1}(1 - \delta_\ell)\delta_{\ell+1} \cdots \delta_{n-1} \rangle$.

- The *n-dimensional Hypercube* $Q(n)$ has vertex-set Z_2^n; the edges of $Q(n)$ connect each string-vertex x with the n strings that differ from x in precisely one bit-position.

[2]For any set S and positive integer k: $Z_k =_{\text{def}} \{0, 1, \cdots, k-1\}$; S^k denotes the set of all length-k strings of elements of S.

1.3. Basic Tools and Facts

A. Gray Codes and Their Transition Sequences

A *length-m d-dimensional Gray code* is a cyclically ordered sequence of m distinct length-d binary words, with every pair of adjacent words differing in precisely one bit-position. Such a sequence clearly specifies an m-vertex cycle in the d-dimensional Hypercube $Q(d)$.

We use *Gray code transition sequences* [10] to construct the Gray codes used here. Let d be the dimensionality of the target Hypercubes, hence, of the desired Gray codes; let $D = \langle d_0, d_1, \cdots, d_t \rangle$ be an ordered sequence of bit-positions (i.e., $d_0 < d_1 < \cdots < d_t \le d$). For any $r \le t$, the rth *Gray code transition sequence specified by* D, denoted $\mathcal{GS}[r; D]$ is the length-$(2^r - 1)$ sequence of integers defined inductively by the following scheme.

$$\begin{aligned} \mathcal{GS}[1; D] &= d_0 \\ \mathcal{GS}[k+1; D] &= \mathcal{GS}[k; D], d_k, \mathcal{GS}[k; D] \end{aligned}$$

We denote by $\mathcal{GS}[r; D]_i$ the ith element of $\mathcal{GS}[r; D]$ (counting, as usual, from 0). For any integer $k \le 2^{r-1}$, one can use $\mathcal{GS}[r; D]$ to construct a length-$2k$ d-dimensional Gray code $\vec{\xi}_0, \vec{\xi}_1, \ldots, \vec{\xi}_{2k-1}$, as follows.

1. Select any length-d binary string as word $\vec{\xi}_0$ of the code.
2. For $0 \le i < k - 1$, generate word $\vec{\xi}_{i+1}$ by flipping bit-position $\mathcal{GS}[r; D]_i$ of $\vec{\xi}_i$.
3. Generate word $\vec{\xi}_k$ by flipping bit-position d of $\vec{\xi}_{k-1}$.
4. For $0 \le i < k - 1$, generate word $\vec{\xi}_{k+i+1}$ by flipping bit-position $\mathcal{GS}[r; D]_i$ of $\vec{\xi}_{k+i}$.

Automatically, word $\vec{\xi}_0$ is obtained by flipping bit-position d of $\vec{\xi}_{2k-1}$. Denote by $\mathcal{GS}[r; D; 2k]$ the sequence of bit-positions flipped in this procedure:

$$\mathcal{GS}[r; D; 2k]_i = \begin{cases} \mathcal{GS}[r; D]_i & \text{if } i \in \{0, 1, \cdots, k-2\} \\ d & \text{if } i \in \{k-1, 2k-1\} \\ \mathcal{GS}[r; D]_{i-k} & \text{if } i \in \{k, k+1, \cdots, 2k-2\} \end{cases}$$

B. Easily Verified Facts

The following easily verified facts are used in developing and/or analyzing our embeddings.

Lemma 1 (a) *Every contiguous subsequence of $\mathcal{GS}[r; D]$ contains at least one element an odd number of times.*
(b) *The just-described procedure generates a length-$2k$ d-dimensional Gray code.*
(c) *The d-dimensional Hypercube $Q(d)$ contains a cycle of every even length $2k$, $2 \le k \le 2^{d-1}$.*
(d) *For all d, the d-dimensional Hypercube $Q(d)$ contains no cycle of odd length.*

2. EMBEDDING THE FFT GRAPH

It is not hard to optimize either dilation or expansion when embedding FFT graphs in Hypercubes; what we accomplish here is to optimize both cost measures simultaneously, and to do so via linear-time algorithms (which specify the vertex-mappings). In fact, we present a family of such algorithms:

Theorem 1 *For each n, $F(n)$ is embeddable as a subgraph of $Q(d_n)$; moreover, one can find the embedding in a* modular *fashion — the embedding of $F(n)$ is an extension of the embedding of $F(n-1)$. All of these embeddings are produced by a linear-time algorithm.*

2.1. The Embedding Strategy

All of the embeddings of $F(n)$ in $Q(d_n)$ that we use to prove the Theorem are specified via two labelling schemes:

- We assign each vertex v of $F(n)$ a unique d_n-bit label $L(v)$, which is its image vertex in $Q(d_n)$.
- We assign each edge (u, v) of $F(n)$ a *bit-position* label $B(u, v) \in \{0, 1, \cdots, d_n - 1\}$ such that $L(u)$ and $L(v)$ differ exactly in bit-position $B(u, v)$.

We simplify our embedding by using a single *bit-position pair* (*bp-pair*, for short) (s_i, c_i) to assign labels to edges between levels $i - 1$ and i of $F(n)$, $1 \leq i \leq n$; all straight-edges between these levels flip[3] bit-position s_i, and all cross-edges between these levels flip bit-position c_i.

> Notes: (a) Edge (u, v) of $F(n)$ maps onto the edge crossing dimension $B(u, v)$ of $Q(d_n)$, between vertices $L(u)$ and $L(v)$; hence the unit dilation of our embeddings.
> (b) Flipping bit-position b corresponds to crossing dimension b of $Q(d_n)$.

Thus, our embedding is specified by means of a *levelled bp-pair sequence* (*LBPS*, for short)

$$S = (s_1, c_1), (s_2, c_2), \ldots, (s_n, c_n).$$

An LBPS almost completely specifies an embedding: When we assign a d_n-bit label $L(v)$ to any single vertex v of $F(n)$, the labels of all remaining vertices are completely determined by the LBPS. We can, and shall, therefore, specify our embeddings by labelling input vertex $v_0 =_{\text{def}} \langle 0, \vec{0} \rangle$ of $F(n)$ with the length-d_n string $\vec{0}$ (thereby assigning it to vertex $\vec{0}$ of $Q(d_n)$) and using an appropriate LBPS to induce the labelling of all other vertices. This strategy reduces the problem of specifying an embedding to the problem of specifying an LBPS $S(n)$ for each FFT graph $F(n)$; and, it reduces the problem of validating a given labelling to the problem

[3]Edge (u, v) of $F(n)$ is said to *flip* bit-position p if $L(u)$ and $L(v)$ differ precisely in bit-position p.

of proving that the label-assignment is one-to-one. This last assertion (about the reduction) is true since any mapping produced by the strategy is *well-defined*, in the sense that the label inductively assigned to each vertex of $F(n)$ is independent of the order of assigning labels. Well-definition is verified by showing that distinct paths from v_0 to any v, which represent distinct inductive label assignments, form cycles in $F(n)$. Since $F(n)$ lacks wraparound, such a cycle must cross each Hypercube dimension an even number of times; hence, each path must assign the same label to v.

The next subsection presents a broad family of embeddings which meet the initial demands of Theorem 1; i.e., they find instances of $F(n)$ as a subgraph of $Q(d_n)$. Subsection 2.3 identifies a subfamily of these embeddings that are modular in the sense of the Theorem. The reader can verify that the embeddings can be specified by linear-time algorithms.

2.2. A Family of Embeddings

Let $\lambda_n = \lceil \log(n+1) \rceil$, and let D be any λ_n-element subset of Z_{d_n}. Define the LBPS

$$S_D(n) = (s_1, c_1), (s_2, c_2), \ldots, (s_n, c_n)$$

as follows:

- $s_\ell = \mathcal{GS}[\lambda_n; D]_{\ell-1}$
- $c_\ell =$ the ℓ^{th} largest integer in the set $Z_{d_n} - D$

for all $\ell \in \{1, 2, \cdots, n\}$. Note the crucial facts that
(a) we assign disjoint bit-positions to straight-edges and cross-edges;
(b) the cross-edges at each level of $F(n)$ are assigned a unique bit-position.

Claim. *For all D, the LBPS $S_D(n)$ specifies an optimal embedding of $F(n)$ in $Q(d_n)$.*

Proof Sketch. Focus on two arbitrary distinct vertices of $F(n)$, $u = \langle \ell_u, \vec{\delta}_u \rangle$ and $v = \langle \ell_v, \vec{\delta}_v \rangle$; say that $\ell_u \geq \ell_v$. Let $u' = \langle n, \vec{\gamma}_u \rangle$ (resp., $v' = \langle 0, \vec{\gamma}_v \rangle$) be the vertex in the *bottom* (resp., the *top*) level of $F(n)$, which is attained by following only *cross-edges* from vertex u (resp., from vertex v). Consider the path P in $F(n)$ that starts at u, follows cross-edges to u', thence follows the *unique* length-n path from u' to v', and finally follows cross-edges to v. Let u'' and v'' be, respectively, the vertices at levels ℓ_u and ℓ_v along the subpath of P that connects u' and v'. We analyze the structure of path P.

The ends of the path. For each level $k \geq \ell_u$ and each level $k < \ell_v$, path P traverses two edges connecting level k with level $k+1$ — one being a cross-edge. If at any level, the other edge is *not* a cross-edge, then clearly $L(u) \neq L(v)$, because $S_D(n)$ assigns each level-k cross-edge a bit-position which is shared by no straight-edge and by no cross-edge at any other level of $F(n)$. Hence, the subpath of P that connects u to u' coincides with the subpath that connects u' to u'', so $u = u''$; similarly, $v = v''$.

The middle of the path. For each remaining level k of $F(n)$, path P traverses only one edge connecting level k to level $k+1$. If this edge were a cross-edge, then it would flip a unique bit-position, thereby assuring that $L(u) \neq L(v)$. Assume, therefore, that all these edges are straight-edges. Since straight-edges on a consecutive sequence of levels of $F(n)$ flip bit-positions that are specified by a contiguous subsequence of a Gray code transition sequence, Lemma 1(a) guarantees that some bit-position is flipped an odd number of times along the middle portion of path P. Once again, this assures that $L(u) \neq L(v)$. □

The just verified Claim establishes the unit dilation, hence, unit congestion of our embeddings; optimality of expansion follows from our choice of λ_n, which ensures that we embed $F(n)$ into $Q(d_n)$. This completes the proof of Theorem 1. □

2.3. A Modular Family of Embeddings

We want to choose an infinite sequence of integers

$$d_0 < d_1 < d_2 < \cdots$$

with the following property. If we define D_n to be the first λ_n elements of the sequence, then for each set D_n, the LBPS S_{D_n} (as specified in Section 2.1) specifies an optimal embedding of $F(n)$ in $Q(d_n)$. The embeddings defined by the sequence of LBPS's S_{D_n} will be the desired *modular* family of optimal embeddings.

Claim. *The infinite sequence of integers defined by*

$$d_k = 2^k + k - 1$$

yields the desired sequence of LBPS's.

Proof Sketch. Our construction of LBPS's always assigns to each cross-edge bit-position c_i a dimension that is *new*, i.e., used for no edge at a lower numbered level than i; in contrast, bit-positions used for straight-edges are reused. However, a *new* dimension must be introduced for straight-edges at least at every level of the form 2^k, since a set D of dimensions can be used for only $2^{|D|}-1$ levels of an LBPS if the induced embedding is to be injective. In other words, a set D leads to an optimal embedding of $F(2^{|D|}-2)$; but D must be augmented if a bigger FFT is to be embedded, by adding to it a dimension which is *new*. To wit, the dimensionality of the smallest Hypercube that will hold an FFT graph usually increases by 1 when we expand $F(n)$ to $F(n+1)$ (which is why just a *new* c_i suffices); but when n is a power of 2, the dimensionality increases by 2 — so bit-position s_i must be *new* also. In order to minimize expansion, we add the smallest as-yet-unused bit-position, namely, $2^{|D|}+|D|-1$ to D. (Straight-edges have consumed $|D|-1$ bit-positions; cross-edges have consumed $2^{|D|}-1$ bit-positions.) □

3. EMBEDDING THE BUTTERFLY AND CCC

Theorem 2 *Every order-n Butterfly graph or CCC graph is embeddable in a Hypercube with* *unit congestion, with optimal expansion, and with dilation* $1 + (n \bmod 2)$. *These embeddings are optimal in all three cost measures and are computable in linear time.*

We establish the upper bounds of Theorem 2 in the next two subsections. Lemma 1(d) establishes optimality of dilation.

Since $B(n)$ and $C(n)$ are smaller than $F(n)$, our embeddings in this Section will be into a (sometimes) smaller Hypercube than in Section 2, namely, $Q(\delta_n)$, where $\delta_n =_{\text{def}} n + \lceil \log n \rceil$.

3.1. Embedding The Butterfly Graph

A. The Underlying Embedding of the FFT Graph

Our embedding of $B(n)$ derives from one specific member of our family of embeddings of $F(n)$. Letting $Even(n) =_{\text{def}} n + (n \bmod 2)$, we define the LBPS

$$S_D(n) = (s_1, c_1), (s_2, c_2), \ldots, (s_n, c_n)$$

that specifies the desired embedding by setting $D = Z_{\lambda_n}$ and defining, for $\ell \in \{1, 2, \cdots, n\}$.

- $s_\ell = \mathcal{GS}[\lambda_n; D; Even(n)]_{\ell-1}$
- $c_\ell = \ell + \lambda_n$.

Remarks. (1) Since $\mathcal{GS}[\lambda_n; D; Even(n)]$ uses no integer $> \lambda_n - 1$, we assign disjoint labels to straight-edges and cross-edges. (2) When n is even, the straight-edges of $F(n)$ induce a cycle in $Q(\delta_n)$; when n is odd, all but one adjacent pair of image-strings differ in one bit-position, while the remaining pair differ in two bit-positions.

B. The Embedding

Our embedding views $B(n)$ as two copies of $F(n-1)$, along with edges between each output and its corresponding input in *both* copies of $F(n-1)$.

Our embedding of $B(n)$ reserves dimension δ_n of $Q(\delta_n)$ as special, thereby partitioning $Q(\delta_n)$ into two copies of $Q(\delta_n - 1)$. We use the embedding of Section 3.1A to embed a copy of $F(n-1)$ into the copy of $Q(\delta_n - 1)$ in which every vertex-address has a 0 in bit-position δ_n; let $\vec{\alpha}0$ be the address of the image in $Q(\delta_n)$ of output $\langle n-1, \vec{0}\rangle$ of $F(n-1)$. Next, we embed a copy of $F(n-1)$ into the copy of $Q(\delta_n - 1)$ in which every vertex-address has a 1 in bit-position δ_n, using the same simple embedding, but mapping the origin vertex $\langle 0, \vec{0}\rangle$ of $F(n-1)$ to vertex $\vec{\alpha}1$ of $Q(\delta_n)$ rather than to vertex $\vec{0}$.

It follows from the analyses in Section 2 that our embedding of $B(n)$

- is well-defined and injective

- embeds all edges of $B(n)$ that do not connect outputs of the copies of $F(n-1)$ with inputs as edges of $Q(\delta_n)$.

We need therefore look only at the output-to-input edges.

Consider first the edges $e(\vec{\xi})$ of $B(n)$ that connect outputs $\langle n-1, \vec{\xi}\rangle$ of a copy of $F(n-1)$ with their corresponding input $\langle 0, \vec{\xi}\rangle$ *in that copy*. Our underlying FFT embedding maps each input-to-output path of straight-edges in $F(n-1)$ to a Hypercube cycle of length $Even(n)$. Hence, when n is even, the cycle has length n, so edge $e(\vec{\xi})$ completes a cycle in $Q(\delta_n)$, hence has unit dilation; when n is odd, the cycle has length $n+1$, so edge $e(\vec{\xi})$ must be routed along a length-2 path in order to complete the cycle in $Q(\delta_n)$, engendering dilation 2 (but congestion is still 1).

The edges that connect an output $\langle n-1, \vec{\xi}\rangle$ of one copy of $F(n-1)$ and its corresponding input $\langle 0, \vec{\xi}\rangle$ *in the other copy* are all embedded as Hypercube edges; in fact, we prove in the complete paper that they all cross dimension δ_n of $Q(\delta_n)$. □

3.2. Embedding The CCC Graph

Our efficient embedding of the CCC graph is simplified by the well-known (and easily verified) fact that each Hypercube $Q(c+d)$ is isomorphic to the product graph $Q(c) \times Q(d)$. Hence, we embed $C(n)$ in $Q(\lambda_{n-1}) \times Q(n)$, rather than explicitly in $Q(\delta_n)$.

Note that Lemma 1(c) guarantees the existence in $Q(\lambda_{n-1})$ of a cycle of length $Even(n)$. This cycle implicitly orders $Even(n)$ of the $2^{\lambda_{n-1}}$ copies of $Q(n)$ that are contained in the product graph $Q(\lambda_{n-1}) \times Q(n)$, so we can talk about "the i^{th} copy of $Q(n)$," call it Q_i. To embed $C(n)$ in $Q(\lambda_{n-1}) \times Q(n)$, we assign each vertex $\langle \ell, \vec{\delta}\rangle$ of $C(n)$ to vertex $\vec{\delta}$ of Q_ℓ. This association

1. is one-to-one: it maps distinct vertices of $C(n)$ to distinct vertices of $Q(\lambda_{n-1}) \times Q(n)$;
2. allows one to route each level-edge of $C(n)$ as a single edge of some Q_i;
3. allows one to route each straight-edge of $C(n)$ as a single edge (resp., a path of length 2) in $Q(\lambda_{n-1}) \times Q(n)$ if n is even (resp., if n is odd), for then the straight-edges form length-n cycles (resp., length-$(n+1)$ cycles) in copies of $Q(\lambda_{n-1})$.

Property 1 guarantees a valid embedding of $C(n)$ in $Q(\delta_n)$. Properties 2 and 3 guarantee unit dilation when n is even and dilation 2 when n is odd. □

ACKNOWLEDGMENT. The research of D. S. Greenberg was supported in part by NSF Grant MIP-86-01885. The research of L. S. Heath was supported in part by NSF Grant DCI-85-04308. The research of A. L. Rosenberg was supported in part by NSF Grants DCI-85-04308 and DCI-87-96236.

4. REFERENCES

1. F. Berman and L. Snyder (1984): On mapping parallel algorithms into parallel architectures. *Intl. Conf. on Parallel Processing.*

2. S.N. Bhatt, F.R.K. Chung, F.T. Leighton, A.L. Rosenberg (1988): Efficient embeddings of trees in hypercubes. Typescript, Univ. of Massachusetts. See also, Optimal simulations of tree machines. *27th IEEE Symp. on Foundations of Computer Science* (1986) 274-282.

3. S.N. Bhatt, F.R.K. Chung, J.-W. Hong, F.T. Leighton, A.L. Rosenberg (1988): Optimal simulations by Butterfly networks. *20th ACM Symp. on Theory of Computing,* 192-204.

4. S.N. Bhatt and I. Ipsen (1985): Embedding trees in the hypercube. Tech. Rpt. DCS/RR-443, Yale Univ.

5. S.H. Bokhari (1981): On the mapping problem. *IEEE Trans. Comp., C-30,* 207-214.

6. R.M. Chamberlain (1988): Gray codes, Fast Fourier Transforms and hypercubes. *Parallel Computing 6,* 225-233.

7. T.F. Chan (1986): On Gray code mapping for mesh-FFTs on binary N-cubes. Tech. Rpt. RIACS-86.17, NASA Ames Research Center.

8. L. Johnsson (1985): Basic linear algebra computations on hypercube architectures. Tech. Rpt., Yale Univ.

9. F.P. Preparata and J.E. Vuillemin (1981): The cube-connected cycles: a versatile network for parallel computation. *C. ACM 24,* 300-309.

10. E. M. Reingold, J. Nievergelt, N. Deo (1977): *Combinatorial Algorithms: Theory and Practice.* Prentice-Hall, Englewood Cliffs, NJ.

11. Y. Saad and M.H. Schultz (1988): Topological properties of hypercubes. *IEEE Trans. Comp. 37,* 867-872.