OPTIMAL EMBEDDINGS OF BUTTERFLY-LIKE GRAPHS IN THE HYPERCUBE

David S. Greenberg†, Lenwood S. Heath§,
Arnold L. Rosenberg

Computer and Information Science Department
University of Massachusetts

COINS Technical Report 88-103

†Yale University, New Haven CT
§Virginia Polytechnic Institute, Blacksburg, VA

OPTIMAL EMBEDDINGS OF BUTTERFLY-LIKE GRAPHS IN THE HYPERCUBE

David S. Greenberg
Department of Computer Science
Yale University
New Haven, CT 06520

Lenwood S. Heath
Department of Computer Science
Virginia Polytechnic Institute
Blacksburg, VA 24061

Arnold L. Rosenberg
Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003

December 19, 1988

Abstract

We present optimal embeddings of three genres of butterfly-like graphs in the (boolean) Hypercube; each embedding is specified via a linear-time algorithm. Our first embedding finds an instance of the FFT graph as a subgraph of the smallest Hypercube that is big enough to hold it; thus, we embed the n-level FFT graph, which has $(n+1)2^n$ vertices, in the $(n+\lceil \log_2(n+1) \rceil)$-dimensional Hypercube, with unit dilation. This embedding yields a mapping of the pipelined FFT algorithm on the Hypercube architecture, which is optimal in all resources (time, processor utilization, load balancing, etc.) and which is on-line in the sense that inputs can be added to the Transform even during the computation. Second, we find optimal embeddings of the n-level Butterfly graph and the n-level Cube-Connected Cycles graph, each of which has $n2^n$ vertices, in the $(n + \lceil \log_2 n \rceil)$-dimensional Hypercube. These embeddings, too, have optimal dilation, congestion, and expansion: The dilation is $1 + (n \bmod 2)$, which is best possible. Our embeddings indicate that these two bounded-degree approximations to the Hypercube do not have any communication power that is not already present in the Hypercube.

1. INTRODUCTION

1.1. The Main Results and Motivation

The main results of this paper demonstrate that the (boolean) Hypercube architecture can efficiently simulate butterfly-like communication patterns. We prove that:

1. The FFT graph is a subgraph of the smallest (Boolean) 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. When the Butterfly or CCC has even order, it is embeddable as a subgraph; when it has odd order, our embedding still has unit congestion,[1] but its dilation increases to 2. This increase in dilation is inevitable.

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

The major notions of the efficiency of a graph embedding are enunciated in [15]: *dilation* measures how edges of the guest graph are "stretched" when they are embedded

[1] Technical terms are defined informally in the next paragraph, and formally in Section 1.2.

in the host graph; *congestion* measures how many edges of the guest graph are routed over a single edge of the guest graph (dilation and congestion jointly measure the maximum delay engendered by the embedding); *expansion* measures the fraction of vertices of the host graph that are actually used as homes for vertices of the guest graph; it is one way of measuring the efficiency of utilizing the processors of the host array.

Motivating our first result is the problem of mapping a parallel algorithm onto a processor array. On the one hand, the algorithm has some natural subtask-interdependence structure, engendered by either data dependencies or control dependencies; on the other hand, the array has a fixed processor-intercommunication network. The *mapping problem* is the problem of accommodating the algorithm's intertask dependence structure to the array's interprocessor communication structure. One typically studies the mapping problem by viewing both of the structures of interest as simple undirected graphs and viewing the mapping problem as one of finding an efficient embedding of the algorithm-graph in the array-graph [2, 3, 6]. Our first result studies the mapping problem for the important *Fast Fourier Transform* (*FFT*) algorithm [1, Ch. 7] which is paradigmatic for convolution-based algorithms, in the popular Hypercube architecture [3, 5, 7-9, 12, 16], versions of which have been built by Intel, N-cube, BBN, and Thinking Machines. We present a family of embeddings, each of which finds an instance of the n-level FFT graph $F(n)$, which has $(n+1)2^n$ vertices, in the d_n-dimensional[2] Hypercube $Q(d_n)$, as a subgraph. We show how to construct optimal embeddings that are *modular*, in the sense 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 can be interpreted as mappings of the *pipelined FFT algorithm* onto the Hypercube architecture, which *utilize all resources optimally* (primarily time, as measured by dilation and congestion, and processor utilization, as measured by expansion); moreover, our modular embeddings can be viewed as mappings that are *on-line*, in the sense that (subject to the limitation of the size of the host Hypercube) inputs can be added to the Transform at any time, even after computation has begun. Our mappings provide yet another example of the efficiency of the Hypercube as an interconnection structure, to supplement earlier work that has shown the Hypercube to be an efficient host for the class of divide-and-conquer algorithms [3] and the class of grid-based algorithms [12], as well as for a number of specific algorithms [7, 8].

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 in the embedding represents the interconnection network

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

to be simulated, and the host graph represents the simulating network [4, 5]. Our second result studies the *Butterfly* and *Cube-Connected-Cycles* networks. These networks were invented as bounded-degree approximations to the Hypercube, which could equal its speed on the important class of *ascend-descend* algorithms. (See [13] for a discussion of both the networks and the class of algorithms.) The embeddings that prove our result can be interpreted as showing that the structure of the Butterfly and CCC does not give those networks any communication power that was not already present in the Hypercube.

A result superficially similar to our first result appears in [8]; in that paper, it is shown that each single level of the FFT graph is a spanning subgraph of the Hypercube, whence one can run the FFT algorithm as fast on the Hypercube as on an architecture with the structure of the FFT graph. Our result is materially harder than that of [8], in that we embed the entire FFT graph into the Hypercube at once. In a somewhat similar vein, it is shown in [7] 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 each pair of data 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

The technical vehicle for our investigation is the following notion of graph embedding. Let G and H be simple undirected graphs, having $|G|$ vertices 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* of the embedding is the maximum number of edges of G that are routed over a single edge of H. The *expansion* of the embedding is the ratio $|H|/|G|$. Clearly, no embedding can have better than unit dilation, and such dilation is achievable only if G is a subgraph of H.

> The graph G represents either the algorithm being mapped or the architecture being simulated. In the former case, the vertices of G represent the *tasks* of the algorithm, and the edges of G represent *intertask dependencies*. In the latter case, G, and in both cases, H represent processor arrays: the vertices of these architecture-graphs represent the *processors* of the arrays; the edges represent *interprocessor communication links*. Thus, the dilation

of an embedding measures the delay incurred because of mismatched adjacencies, while the congestion of the embedding measures the contention for the communication links of "processor array" H, either when one executes "algorithm" G on "processor array" H or when one uses "processor array" H to simulate the most costly communication link of "processor array" G. Congestion can be resolved either by increasing the bandwidth of the links, at the cost of increased hardware and increased layout area, or via queuing of messages, at the cost of increased delay; our embeddings have such small congestion that increased bandwidth seems to be the appropriate response. The expansion of an embedding is a measure of how efficiently G utilizes the processors of H.

Our specific focus here is on embeddings of three finite families of graphs $\mathcal{G}_1$, $\mathcal{G}_2$, and $\mathcal{G}_3$ in a fourth finite family $\mathcal{H}$. We seek the best possible embeddings — relative to dilation and congestion — 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 are able here to optimize all three cost measures simultaneously. In [11] examples are presented wherein optimizing either the dilation or the expansion of an embedding forces the other cost to grow without bound.

The graph families of interest to us here are FFT graphs, Butterfly graphs, and CCC graphs (which play the role of our G's) and Hypercubes (which play the role of our H's).

- Let n be a positive integer. The *order-n (2^n-input) FFT graph* $F(n)$ (so named because it reflects the data-dependency structure of the 2^n-input FFT algorithm) has vertex-set[3] $V_n = Z_{n+1} \times Z_2^n$. 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 of $F(n)$ are called *inputs*, and vertices at level n of $F(n)$ are called *outputs* (in deference to the algorithmic origins of the graph). The edges of $F(n)$ are of two types: For each $\ell \in Z_n$ and each $\delta_0\delta_1 \cdots \delta_{n-1} \in Z_2^n$, the 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 $F(n)$

[3] 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.

and is connected by a *cross-edge* with vertex[4]

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

It is often useful to 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 $F(n-1)$, on the one side, to the k^{th} and $(k + 2^{n-1})^{\text{th}}$ new outputs, on the other side. Thus, $F(n)$ has $(n+1)2^n$ vertices and $n2^{n+1}$ edges.

- Let n be a positive integer. The *order-n Butterfly graph* $B(n)$ is the FFT graph $F(n)$, with "wraparound" obtained by identifying each input vertex $\langle 0,\ \vec{\delta} \rangle$ with the corresponding output vertex $\langle n,\ \vec{\delta} \rangle$. Thus, $B(n)$ has $n2^n$ vertices and $n2^{n+1}$ edges.

- Let n be a positive integer. The *order-n Cube-Connected Cycles (CCC) graph* $C(n)$ has vertex-set $W_n = Z_n \times Z_2^n$. For each vertex $v = \langle \ell,\ \vec{\delta} \rangle \in W_n$, we call ℓ the *level* of v and $\vec{\delta}$ the *position-within-level (PWL) string* of v. The edges of $C(n)$ are of two types: For each $\ell \in Z_n$ and each $\delta_0\delta_1 \cdots \delta_{n-1} \in Z_2^n$, the vertex

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

  is connected by a *straight-edge* with vertex

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

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

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

- Let d be a nonnegative integer. The *d-dimensional (boolean) Hypercube* $Q(d)$ has vertex-set Z_2^d; the edges of $Q(d)$ connect each string-vertex x with the d strings that differ from x in precisely one bit-position. Thus, $Q(d)$ has 2^d vertices and $d2^{d-1}$ edges.

1.3. Basic Tools and Facts

A. Gray Codes and Their Transition Sequences

[4] $\oplus$ denotes addition modulo 2.

Gray codes and sequences that generate them are the main technical devices in our study.

A *length-m d-dimensional Gray code* is a cyclically ordered sequence of m distinct length-d binary words, having the property that words adjacent in the sequence differ in precisely one bit-position. In the obvious way, such a Gray code can be viewed as (specifying) an m-vertex cycle in the d-dimensional Hypercube $Q(d)$.

Many devices are known for constructing Gray codes of desired lengths and dimensionalities. The device used here is the following notion of *Gray code transition sequence* [14]. Let d be a positive integer (which represents the dimensionality of the target Hypercubes, hence of the desired Gray codes), and let $D = \langle d_0, d_1, \cdots, d_t \rangle$ be an ordered sequence of bit-positions, i.e., integers, each $\leq d$; symbolically,

$$d_0 < d_1 < \cdots < d_t \leq d.$$

For any $r \leq t$, the r^{th} *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 i^{th} element of $\mathcal{GS}[r; D]$ (counting, as usual, from 0). For any integer $k \leq 2^{r-1}$, 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 $0 \leq i < k - 1$, 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 $0 \leq i < k - 1$, generate word $\vec{\xi}_{k+i+1}$ by flipping bit-position $\mathcal{GS}[r; D]_i$ of $\vec{\xi}_{k+i}$.

It follows automatically that word $\vec{\xi}_0$ is obtained by flipping bit-position d of $\vec{\xi}_{2k-1}$. Let us 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

Having developed the main technical devices needed for our embeddings, we present a number of easily verified facts about our formal framework. We leave proofs to the reader.

Since every two occurrences of an integer h in $\mathcal{GS}[r; D]$ are separated by an occurrence of some integer $\geq h + 1$, it follows that

Lemma 1 *Every contiguous subsequence of $\mathcal{GS}[r; D]$ contains at least one element an odd number of times.*

Using Lemma 1 on prefixes of $\mathcal{GS}[r; D]$, one verifies that

Lemma 2 *The just-described procedure generates a length-$2k$ d-dimensional Gray code.*

The following immediate consequence of Lemma 2 is useful in Section 3.

Lemma 3 *For all d, the d-dimensional Hypercube $Q(d)$ contains a cycle of every even length $4 \leq 2k \leq 2^d$.*

Since $Q(d)$ is bipartite, the parity requirement in Lemma 3 cannot be removed:

Lemma 4 *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 find a unit-dilation embedding of $F(n)$ in $Q(2n)$, by assigning two new dimensions for each level of $F(n)$; but this embedding has expansion $\Omega(2^n/n)$. Likewise, it is not hard to find a dilation-2 (expansion-optimal) embedding of $F(n)$ in $Q(d_n)$, which is the smallest Hypercube that holds $F(n)$, using embedding techniques analogous to those used in [5]. 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, all based on Gray codes and Gray code transition sequences. Stated formally, we prove

Theorem 1 *Every FFT graph is embeddable in a Hypercube, with unit dilation and congestion, and with optimal expansion. Thus, for each n, $F(n)$ is a subgraph of $Q(d_n)$; moreover, there is a family of such embeddings that is* modular *in the sense that the embedding of $F(n+1)$ is an extension of the embedding of $F(n)$. 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[5] bit-position s_i, and all cross-edges between these levels flip bit-position c_i.

> Note that edge (u, v) of $F(n)$ is mapped by our embeddings onto the edge crossing dimension $B(u, v)$ of $Q(d_n)$, between vertex $L(u)$ and vertex $L(v)$ (whence the unit dilation of our embeddings). Note also that 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)

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

One verifies easily that the LBPS gives us almost all the information we need to specify the embedding completely: 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 embedding 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 $Q(d_n)$ in the embedding) 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 — i.e., verifying that it actually specifies an embedding — to the problem 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 inductively assigning labels. Well-definition is verified as follows.

[5] 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.

Proposition 1 *Any mapping of the vertices of the FFT graph to the vertices of the Hypercube which is induced by an LBPS is well-defined and has unit dilation.*

Proof. The unit-dilation property being immediate from the fact that the mapping flips just one bit-position for each edge of $F(n)$, let us concentrate on verifying the well-definition of the labelling procedure.

Assume that vertex v is assigned label $L(v)$ when the labelling is induced by the path P from vertex v_0 to v and that it is assigned label $L'(v)$ when the labelling is induced by the path P' from vertex v_0 to v. We claim that $L(v) = L'(v)$.

We begin with three basic facts about cycles in $F(n)$.

1. For each level $\ell \in \{0, 1, \cdots, n\}$, the number of level-$\ell$ edges in any cycle in $F(n)$ is even.

2. For each level $\ell \in \{0, 1, \cdots, n\}$, the number of level-$\ell$ cross-edges in any cycle in $F(n)$ is even.

3. For each level $\ell \in \{0, 1, \cdots, n\}$, the number of level-$\ell$ straight-edges in any cycle in $F(n)$ is even.

The first fact follows since there is no "wraparound" in $F(n)$, so any cycle must re-cross levels. The second fact follows since every level-ℓ cross-edge flips bit-position ℓ of the current vertex's PWL string, and no straight-edge flips a bit; therefore, in order to regain a previous vertex, one must restore every flipped bit-position by re-crossing level ℓ via a cross-edge. The third fact follows from the first two via arithmetic.

Next, note that since we are dealing here only with labelling schemes that are induced by LBPS's, crossing a level of $F(n)$ twice using the same type of edge — a cross-edge or a straight-edge — flips the same bit-position of the Hypercube label twice, hence leaves the label unchanged.

Finally, consider the cycle formed by tracing path P from vertex v_0 to vertex v, followed by tracing the reverse of path P' from vertex v to vertex v_0. By the foregoing facts, each Hypercube dimension appears an even number of times around the cycle; hence, the parity of the number of appearances of each dimension on P must be the same as the corresponding parity on P'. It follows that $L(v) = L'(v)$. □

The next subsection is devoted to specifying and validating, within this simplified framework, 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. We leave to the reader the straightforward verification that all of the embeddings presented can be produced by linear-time algorithms.

2.2. A Family of Embeddings

A. The Embeddings

Let $\lambda_n = \lfloor \log(n+1) \rfloor$, and let D be any λ_n-element subset of Z_{d_n}. (Note that $\lambda_n = d_n - n$.) 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) the set of bit-positions we assign to the straight-edges of $F(n)$ is disjoint from the set of labels we assign to the cross-edges.
(b) The cross-edges at each level of $F(n)$ are assigned a unique bit-position. (This is possible only because $\lambda_n \leq d_n - n$.)

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

B. Validation

Our proof that the LBPS $S_D(n)$ labels $F(n)$ injectively, hence specifies an embedding, depends on Lemma 1 and on the following easily verified property of FFT graphs.

Lemma 5 *For all PWL strings $\vec{\gamma}, \vec{\delta} \in \{0,1\}^n$, there is a unique length-n path in $F(n)$ connecting vertex $\langle 0, \vec{\gamma} \rangle$ and vertex $\langle n, \vec{\delta} \rangle$.*

Proof. Level-ℓ edges of $F(n)$ are the only ones that affect bit-position ℓ of vertices' PWL strings. A length-n path that connects a vertex at level 0 with a vertex of level n traverses each level of $F(n)$ precisely once. Therefore, any path that connects vertices $\langle 0, \vec{\gamma} \rangle$ and $\langle n, \vec{\delta} \rangle$ must traverse the straight-edge between levels i and $i+1$ for every

bit-position i in which the PWL strings $\vec{\gamma}$ and $\vec{\delta}$ agree, and it must traverse the cross-edge between levels i and $i+1$ for every bit-position i in which the PWL strings $\vec{\gamma}$ and $\vec{\delta}$ differ. □

Now, we proceed with the case analysis that establishes the injectiveness of the labelling produced by $S_D(n)$.

Focus on two arbitrary distinct vertices of $F(n)$, say $u = \langle \ell_u, \vec{\delta}_u \rangle$ and $v = \langle \ell_v, \vec{\delta}_v \rangle$. Without loss of generality, assume that $\ell_u \geq \ell_v$. Let $u' = \langle n, \vec{\gamma}_u \rangle$ be the vertex in the *bottom* level of $F(n)$, that is attained by following only *cross-edges* from vertex u; let $v' = \langle 0, \vec{\gamma}_v \rangle$ be the vertex in the *top* level of $F(n)$ that is attained by following only *cross-edges* from vertex v. Consider the path P in $F(n)$ that starts at u, traverses cross-edges until it reaches u', thence follows the unique length-n path from u' to v' (cf. Lemma 5), and finally traverses cross-edges to end up at 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 now analyze the structure of the path P.

The ends of the path. For each level $k \geq \ell_u$ and each level $k < \ell_v$, the path P traverses two edges connecting level k with level $k+1$ — and one of these edges is a cross-edge. If at any of these levels, the other edge is *not* a cross-edge, then it is immediate that $L(u) \neq L(v)$. To wit, the LBPS $S_D(n)$ assigns each level-k cross-edge a bit-position that is shared by no straight-edge and by no cross-edge at any other level of $F(n)$; hence, the net effect of traversing the two edges would be to flip some bit-position of $L(u)$ that is flipped nowhere else. We conclude that the subpath of P that connects u to u' coincides with the subpath that connects u' to u'', which means that $u = u''$. By similar reasoning, $v = v''$.

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

We have exhausted all possible structures for the path P and shown that each possibility guarantees that $L(u) \neq L(v)$. Since u and v were arbitrary, we have shown that the mapping induced by the LBPS $S_D(n)$ is injective. □

Proposition 1, together with the just verified Claim, establish that we have spec-

ified a family of unit-dilation (hence, unit-congestion) embeddings. The expansion-optimality of the embeddings follows from our choice of the parameter λ_n, which ensures that our embedding maps $F(n)$ into $Q(d_n)$, the smallest Hypercube big enough to hold it. This completes the proof of Theorem 1. □

2.3. A Family of Modular 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.

A. Validation

When we construct LBPS's from sets D of dimensions in Section 2.1, we always assign to each cross-edge bit-position c_i a dimension that is *new* in the sense that it is used for no edge at a lower numbered level than i. In contrast, the bit-positions used for straight-edges are reused continually; but a *new* dimension must be introduced at least at every level whose index is a power of 2, since any given set D of dimensions can be used for only $2^{|D|} - 1$ levels of an LBPS (if the induced embedding is to be injective). This means that a given set of dimensions D leads to an optimal embedding of an order-$(2^{|D|} - 2)$ FFT graph in a Hypercube, and D must be augmented if a bigger FFT is to be embedded. When we augment D, we must add to it a dimension that is *new*. To see this, note that the dimensionality of the smallest Hypergraph 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* cross-edge bit-position c_i suffices); but when n is a power of 2, then the dimensionality of the target Hypercube increases by 2, so we need the bit-position s_i to be *new* also. In order to retain optimal expansion, we want to augment D with the smallest as-yet-unused bit-position. We see easily that this smallest number is $2^{|D|} + |D| - 1$; to wit, straight-edges have consumed $|D| - 1$ bit-positions, while cross-edges have consumed $2^{|D|} - 1$ bit-positions. The claim follows. □

2.4. More on Modular Embeddings

The embeddings we have presented thus far all derive from Gray codes and their transition sequences. Other families of equally efficient embeddings exist. To illustrate this point, we present, without validation or analysis, the (historically) first family of embeddings of FFT graphs as subgraphs of Hypercubes we found in the course of the current research. The somewhat lengthy validation of these embeddings can be found in [10].

Call an LBPS $S = (s_1, c_1), (s_2, c_2), \ldots, (s_n, c_n)$ *proper* if, for each i, at least one of s_i or c_i is *new*. We now construct, for every n, a proper LBPS $S(n)$ that specifies an embedding in stages, ensuring propriety at every stage.

Partition the levels $\{0, 1, \cdots, n\}$ of $F(n)$ into *tiers*, tier k being the set of levels

$$\{i : 2^k \leq i \leq 2^{k+1} - 1\} \cap Z_{n+1}.$$

Let the singleton $\{0\}$ constitute tier -1.

In common with our other modular embeddings:

- we obtain the LBPS $S(n+1)$ from the LBPS $S(n)$;
- we always use any *new* bit-position as one becomes available.

As we noted earlier, increasing the order of our embedded FFT graph always ensures the existence of at least one *new* bit-position to use in $S(n+1)$. Let us always use this *new* bit-position to label the straight-edge at level n, i.e., to be bit-position s_n. When $n = 2^k$ is a power of 2, two *new* bit-positions are available for the expanded labelling when we proceed to $F(n+1)$. In this case, we call the pair (s_n, c_n) of *new* bit-positions *shield positions* for tier k of all $F(m)$, $m \geq n$. Given this strategy, we can specify explicitly

- $s_\ell = \ell + \lfloor \log \ell \rfloor$ for all $\ell \in \{1, 2, \cdots, n\}$;
- $c_{2^k} = 2^k + k + 1$ for all $k \in \{1, 2, \cdots, \lfloor \log n \rfloor\}$.[6]

Finally, we specify the c_i's that are not shield positions. We proceed inductively, based on the tier number k, the case $k = 1$ being trivial. Having chosen the c_i's for tier $k-1$, we choose the $2^k - 1$ c_i's for tier k as follows.

[6] Our specification is consistent because $2^k + k + 1 = \ell + \lfloor \log \ell \rfloor + 1$ when $\ell = 2^k$.

- $c_{2^k+i} = \begin{cases} s_{2^{k-1}+i} & \text{if } 1 \le i < 2^{k-1} \\ s_{2^{k-1}} & \text{if } i = 2^{k-1} \\ c_{2^{k-1}+i} & \text{if } 2^{k-1} < i < 2^k \end{cases}$

3. EMBEDDING THE BUTTERFLY AND CCC

The Butterfly graph, whose structure so closely approximates that of the FFT graph, can be embedded in the Hypercube almost as efficiently as can the FFT graph (via a similar, but still quite distinct mapping). Since the structure of the CCC graph is even closer to that of the Hypercube than is the structure of either the Butterfly graph or the FFT graph, it yields to an even simpler mapping technique to obtain an embedding in the Hypercube whose efficiency is identical to that of our embedding of the Butterfly. We begin by stating our second result formally.

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.*

The embeddings that establish the upper bounds of Theorem 2 are presented in the next two subsections. The lower bound implicit in the last sentence of the Theorem follows immediately from Lemma 4.

Since the Butterfly and CCC graphs have fewer vertices than does the FFT graph of the same order, our embeddings in this Section will be into a (sometimes) smaller Hypercube than we used in Section 2. Specifically, in this Section, we shall embed the order-n Butterfly graph $B(n)$ and the order-n CCC graph $C(n)$ into the δ_n-dimensional Hypercube $Q(d_n)$, where $\delta_n =_{\text{def}} n + \lfloor \log n \rfloor$.

3.1. Embedding The Butterfly Graph

A. The Underlying Embedding of the FFT Graph

Our embedding of the Butterfly derives from one specific member of the family of embeddings of the FFT graph described in Section 2.2. 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

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

for all $\ell \in \{1, 2, \cdots, n\}$.

Remarks. (1) Since $\mathcal{GS}[\lambda_n; D; Even(n)]$ uses no integer greater than $\lambda_n - 1$, the labels we assign to the straight-edges of $F(n)$ are disjoint from the labels we assign to the cross-edges.
(2) When n is even, the images of the straight-edges of $F(n)$ form 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 Underlying View of the Butterfly

Our embedding employs the following characterization of Butterfly graphs. The order-n Butterfly $B(n)$ consists of two copies of the order-$(n-1)$ FFT graph $F(n-1)$, along with edges between each output and its corresponding input in *both* copies of $F(n-1)$. This characterization should be compared to that of $F(n)$ as consisting of two copies of $F(n-1)$ and two sequences of 2^{n-1} new vertices (which will become the outputs for $F(n)$), along with edges between each output in each copy of $F(n-1)$ and its corresponding vertex in *both* sequences of new vertices.

C. The Embedding

We begin specifying our embedding of $B(n)$ by reserving the highest dimension, δ_n, of $Q(\delta_n)$ as special, thereby partitioning $Q(\delta_n)$ into two subcubes, each a copy of $Q(\delta_n - 1)$. We embed one 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, using the simple embedding of Section 3.1A; let $\vec{\alpha}0$ be the address of the vertex of $Q(\delta_n)$ to which the leftmost output, $\langle n-1, \vec{0}\rangle$ of this copy of $F(n-1)$ is mapped. Next, we embed the second 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 strategy, 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 $\vec{0}$.

D. Validation

We validate our embedding in two steps: We note first those properties (injectiveness and costs) that are inherited from the embeddings of $F(n-1)$ into the subcubes; we then consider the dilations and congestions of the edges that connect the outputs of the copies of $F(n-1)$ with their corresponding inputs in both copies.

Hereditary properties. It follows from the analyses in Sections 2.1 and 2.2 that our embedding of $B(n)$

- is well-defined and injective (It is induced by an LBPS, and it embeds the two copies of $F(n-1)$ disjointly.)
- embeds all edges of $B(n)$ that do not connect outputs of the copies of $F(n-1)$ with inputs as edges of $Q(\delta_n)$ (It uses the paradigm of Section 2.2.)

The new edges. Consider first the edges of $B(n)$ that connect an output $\langle n-1, \vec{\xi}\rangle$ of a copy of $F(n-1)$ with its corresponding input $\langle 0, \vec{\xi}\rangle$ *in that copy.* As we remarked in Section 3.1A, 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 each output-to-input edge we are considering here completes the cycle in $Q(\delta_n)$, hence has unit dilation. When n is odd the cycle has length $n+1$, so each output-to-input edge we are considering here must be mapped onto a length-2 Hypercube path in order to complete the cycle in $Q(\delta_n)$, engendering dilation 2; congestion is still 1, since the length-2 paths are used only to close the cycles.

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, hence are all embedded with unit dilation and congestion; in fact, they all cross dimension δ_n of $Q(\delta_n)$. We now verify this assertion.

Let us look first at the edges that connect *outputs of the first copy* of $F(n-1)$ with *inputs of the second copy.* Note that the image $\vec{\alpha}0$ of output $\langle n-1, \vec{0}\rangle$ of the first copy of $F(n-1)$ is adjacent in $Q(\delta_n)$ to the image $\vec{\alpha}1$ of input $\langle 0, \vec{0}\rangle$ of the second copy of $F(n-1)$, across dimension δ_n. Let $\langle n-1, \vec{\xi}\rangle$ be any output of the first copy of $F(n-1)$. Consider the cycle in $Q(\delta_n)$ that

1. starts at vertex $\vec{\alpha}0$ (the image of output $\langle n-1, \vec{0}\rangle$ of the first copy of $F(n-1)$),
2. follows the image of the unique length-$(n-1)$ path in the first copy of $F(n-1)$, to input $\langle 0, \vec{\xi}\rangle$,
3. crosses to the image of output $\langle n-1, \vec{\xi}\rangle$ of the second copy of $F(n-1)$,
4. continues along the image of the unique path in the second copy of $F(n-1)$ to input $\langle 0, \vec{0}\rangle$,
5. completes the cycle by crossing to vertex $\vec{\alpha}0$;

By design, the last step of this cycle has just to cross dimension δ_n of $Q(\delta_n)$. Further, the images of the two unique length-$(n-1)$ paths in the cycle must cross *exactly* the same Hypercube dimensions, the same numbers of times. This is true since both copies

are embedded using the same LBPS; hence the only way the described cycle could cross every Hypercube edge that it uses twice is for these paths either both to follow a cross-edge at a given level or both to follow a straight-edge. Thus, since we have described a *cycle* in $Q(\delta_n)$, the one step we have not yet mentioned — that crosses from the image of input $\langle 0, \vec{\xi} \rangle$ of the first copy of $F(n-1)$ to the image of output $\langle n-1, \vec{\xi} \rangle$ of the second copy, must also cross dimension δ_n. Since the PWL $\vec{\xi}$ was arbitrary, we have thus proved that the image of every output of the second copy of $F(n-1)$ is adjacent across dimension δ_n to the image of the corresponding input of the first copy.

We look finally at the edges that connect *outputs of the second copy* of $F(n-1)$ with *inputs of the first copy*. Note first that the image of output $\langle n-1, \vec{0} \rangle$ of the second copy of $F(n-1)$ is adjacent across dimension δ_n to input $\langle 0, \vec{0} \rangle$ of the first copy. We can use an argument symmetric to that of the preceding paragraph to prove that all of the edges we consider here cross dimension δ_n, thereby completing the proof of the claim, hence of the Butterfly portion of Theorem 2. To this end, note that the image in $Q(\delta_n)$ of the unique path in the first copy of $F(n-1)$ from output $\langle n-1, \vec{0} \rangle$ to input $\langle 0, \vec{0} \rangle$ crosses just the same Hypercube dimensions the same numbers of times as does the image of the unique path in the second copy of $F(n-1)$ from input $\langle 0, \vec{0} \rangle$ to output $\langle n-1, \vec{0} \rangle$. Details are left to the reader. □

3.2. Embedding The CCC Graph

Our efficient embedding of the CCC graph is simplified by the following well-known (and easily verified) fact about the Hypercube.

Lemma 6 *For all integers c and d, the Hypercube $Q(c+d)$ is isomorphic to the product graph $Q(c) \times Q(d)$.*[7]

Lemma 6 allows us to embed $C(n)$ in the product graph $Q(\lceil \log n \rceil) \times Q(n)$, rather than explicitly in $Q(n + \lceil \log n \rceil)$.

To achieve this embedding, note that Lemma 3 guarantees the existence in $Q(\lceil \log n \rceil)$ of a cycle of length $Even(n)$. This cycle implicitly orders $Even(n)$ of the $2^{\lceil \log n \rceil}$ copies of $Q(n)$ that are contained in the product graph $Q(\lceil \log n \rceil) \times Q(n)$, so we can freely talk about "the i^{th} copy of $Q(n)$," call it Q_i. To embed $C(n)$ in $Q(\lceil \log n \rceil) \times Q(n)$, we assign each vertex $\langle \ell, \vec{\delta} \rangle$ of $C(n)$ to vertex $\vec{\delta}$ of Q_ℓ. This association has the following properties.

[7] For graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$, the *product* graph $G \times H$ has vertex-set $V_G \times V_H$. Let u and v be vertices of G, and let x and y be vertices of H. Then $(\langle u, x \rangle, \langle v, x \rangle)$ is an edge of $G \times H$ just when (u, v) is an edge of G; $(\langle u, x \rangle, \langle u, y \rangle)$ is an edge of $G \times H$ just when (x, y) is an edge of H.

1. The association is one-to-one, in the sense that it maps distinct vertices of $C(n)$ to distinct vertices of $Q(\lceil \log n \rceil) \times Q(n)$.

2. Under this association, each level-edge of $C(n)$ can be realized via a single edge of one of the copies of $Q(n)$. (Indeed, $C(n)$ was invented to have the property that the level-edges at each level ℓ "mimic" the edges across dimension ℓ of $Q(n)$ [13].)

3. Under this association, the images of the endpoints of each straight-edge of $C(n)$ that goes from level ℓ to level $\ell + 1 \pmod n$ are adjacent if n is even, for then the straight-edges form length-n cycles in copies of $Q(\lfloor \log n \rfloor)$; the images are at distance at most 2 if n is odd, for then the straight-edges form length-$(n + 1)$ cycles in copies of $Q(\lceil \log n \rceil)$.

Property 1 assures us that we have described a valid embedding of $C(n)$ in $Q(\delta_n)$. Properties 2 and 3 assure us that the dilation of the embedding is 1 when n is even and is 2 when n is odd: Level-edges of $C(n)$ can be routed as unit-length paths in $Q(\delta_n)$, while straight-edges can be routed as paths of length at most 2. This completes the proof. □

4. A REMAINING CHALLENGE

Our results lend yet more evidence of the power of the Hypercube network. Now that we know that Hypercubes can simulate butterfly-like networks efficiently, the foremost unresolved question about the Hypercube's power is the issue of how efficiently it can simulate shuffle-like networks, such as the deBruijn and Shuffle-Exchange networks. Resolving this issue is an inviting challenge.

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.

5. REFERENCES

1. A.V. Aho, J.E. Hopcroft, J.D. Ullman (1974): *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA.

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

3. 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.

4. 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.

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

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

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

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

9. T.C. Chen, M.D.F. Schlag, C.K. Wong (1983): The hypercube connection network. IBM Report RC-10219.

10. D.S. Greenberg, L.S. Heath and A.L. Rosenberg (1988): Optimal embeddings of FFT graphs in the Hypercube. Tech. Rpt. 88-23, Univ. Massachusetts.

11. J.-W. Hong, K. Mehlhorn, A.L. Rosenberg (1983): Cost tradeoffs in graph embeddings. *J. ACM 30,* 709-728.

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

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

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

15. A.L. Rosenberg (1981): Issues in the study of graph embeddings. In *Graph-Theoretic Concepts in Computer Science: Proceedings of the International Workshop WG80*, Bad Honnef, Germany (H. Noltemeier, ed.) *Lecture Notes in Computer Science 100*, Springer-Verlag, New York 150-176.

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