Optimal Simulations of Tree Machines

Sandeep Bhatt
Fan Chung
Tom Leighton
Arnold Rosenberg

YALEU/DCS/TR-495
September 1986

Optimal Simulations of Tree Machines

(Preliminary Version)

Sandeep N. Bhatt[1]
Fan R. K. Chung[2]
F. T. Leighton[3]
Arnold L. Rosenberg[4]

Abstract—

Universal networks offer the advantage that they can execute programs written for simpler architectures without significant run-time overhead. In this paper we investigate simulations of tree machines; the fact that divide-and-conquer algorithms are programmed naturally on trees motivates our investigation.

Among various proposals for parallel computing the boolean hypercube has emerged as a particularly versatile network. It is well known that programs for multi-dimensional grid machines, for example, can be executed on a hypercube with no communications overhead by embedding the grid as a subgraph of the hypercube. Our first result is that a program for any tree machine can be executed on the hypercube with constant overhead. More precisely, every cycle of a synchronous binary tree can be simulated in $O(1)$ cycles on a hypercube, independent of the shape of the tree. The algorithm to embed the tree within the hypercube runs in polynomial time. We also give efficient simulations of arbitrary binary trees on the complete binary tree, the FFT and shuffle–exchange networks.

1 Introduction

A number of supercomputer architectures interconnecting hundreds or thousands of processors have been proposed in recent years. Prominent is the boolean hypercube, different versions of which have been built at Intel, N-cube, BBN, and Thinking Machines. The hypercube offers a rich interconnection topology with

[1]Department of Computer Science, Yale University, New Haven CT 06520.
[2]Bell Communications Research, Morristown NJ 07960.
[3]Department of Mathematics, M.I.T., Cambridge, MA 02139.
[4]Department of Computer Science, University of Massachusetts, Amherst MA 01003.

high communication bandwidth, low diameter, and a recursive structure naturally suited to divide-and-conquer applications. More importantly, the hypercube supports efficient routing algorithms and can therefore simulate any realistic parallel machine efficiently. Using Batcher's deterministic sorting scheme or Valiant's randomized message routing algorithm for instance, the hypercube can simulate a PRAM, and hence any realistic parallel machine with only a small polylogarithmic multiplicative increase in time. This universality property makes the hypercube extremely attractive for parallel computing.

Many parallel architectures can be simulated on the hypercube without the logarithmic increase in time. For example, every $2^{d_1} \times \cdots \times 2^{d_n}$ grid is a subgraph of the $2^{\sum d_i}$ node hypercube. Such multi-dimensional grids can therefore be simulated on a hypercube with no communications overhead. The importance of grid algorithms to scientific applications coupled with the capability of a hypercube to simulate grids of different aspect ratios has often been cited as an important consideration in building hypercubes. Johnsson [12] gives a survey of various efficient matrix algorithms on the hypercube.

What other networks can be simulated on a hypercube with little or no communications overhead? This question remains largely unexplored. In this paper we take the first step to investigate simulations of binary trees within the hypercube. Highly parallel divide-and-conquer algorithms can be conveniently programmed on an abstract binary tree machine, as can concurrent data structures [1, 9]. While the complete binary tree is suitable for a number of applications, there are instances when the divide-and-conquer tree is not complete. For example, in finite-element computations the tree generated by recursively decomposing a region into smaller subregions is, in general, neither complete nor binary. This paper considers "static" simulations only; we assume that the binary tree is fixed in size and shape and does not evolve in time. We also assume that all nodes of the tree (and not just the leaves) are active simultaneously, as is the case when different computations are pipelined through a single fixed tree. Our main result is that the hypercube can simulate every binary tree with only a small constant factor overhead in communications cost. This improves results of Bhatt and Ipsen [4] who give a simulation with communications overhead $\log \log N + O(1)$.

For many years graph theorists have been interested in constructions of universal graphs for important families of graphs. Chung and Graham [5] briefly describe some of the early work in this area. The archetypical problem is: given a class Ψ of graphs on N nodes, construct a universal graph H on N nodes with the fewest edges necessary so that every graph G in Ψ is a subgraph of H. The subgraph property is attractive in the context of parallel computing because it implies no communication overhead in simulating any graph in Ψ.

The case when Ψ is the set of N node trees has received considerable attention. Following [6, 7], Chung and Graham [8] constructed a universal graph for trees with $O(N \log N)$ edges, which is optimal up to constant factors for trees with unbounded degree [5]. For the case when Ψ is the class of trees of bounded degree, we give a bounded–degree universal graph. Since our graph has bounded–degree, it has a linear number of edges, thus improving the previous bound for arbitrary trees. Friedman and Pippenger [10] have recently shown that an expanding graph with N nodes and $O(N)$ edges is universal for binary trees with αN nodes $(0 < \alpha < 1)$.

The remainder of this paper is organized as follows. Section 2 gives definitions and illustrates simple embeddings of binary trees within the hypercube. Section 3 sketches the combinatorial argument basic to our embedding technique of Section 4 in which we describe how to simulate any binary tree on the hypercube with constant communication overhead. Section 5 describes the new construction of a bounded–degree universal graph for trees. Section 6 concludes with a number of extensions and open questions.

2 Definitions

The problem of simulating one network by another is modeled as a *graph embedding* problem. An embedding $< \phi, \rho >$ of a graph $G = (V_G, E_G)$ into a graph $H = (V_H, E_H)$ is defined by an injective mapping from V_G to V_H, together with a mapping ρ that maps $(u, v) \in E_G$ onto a path $\rho(\phi(u), \phi(v))$ in H that connects $\phi(u)$ and $\phi(v)$. The *dilation* of the edge (u, v) under $< \phi, \rho >$ equals the length of the path $\rho(\phi(u), \phi(v))$ in H. The *dilation* of an embedding $< \phi, \rho >$ is the maximum dilation, over all edges in G, under $< \phi, \rho >$.

We measure the quality of an embedding with three cost functions — *expansion, dilation,* and *load-factor*. Following Rosenberg [13], define the *expansion* of an embedding $< \phi, \rho >$ of G into H to be the ratio of the size of V_H to the size of V_G. Intuitively, expansion measures processor utilization. The *load-factor* $\lambda(e)$ of an edge e in H is the number of paths that pass through e which are images of edges in G, i.e., $\lambda(e_H) = |\{e \in E_G : \rho(e) \text{ contains } e_H\}|$, and the load-factor λ of an embedding is defined to be the maximum load-factor over all edges in H.

Our model of synchronous parallel networks assumes that a processor (node of G or H) can communicate with each of its neighbors in one clock cycle, so that edges serve as bidirectional links. We restrict attention to simulations in which each cycle of G is simulated by a series of cycles of H, before the simulation of the next clock cycle of G is begun. For an embedding $< \phi, \rho >$ of G, each communication across an edge in $e \in E_G$ is effected by transmitting the message

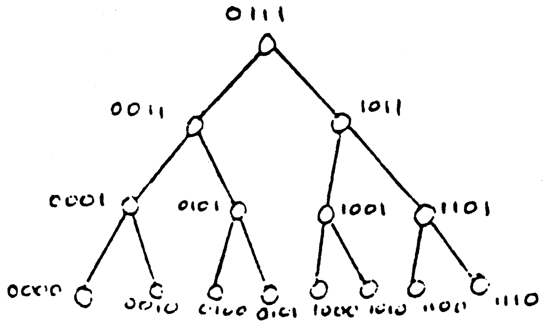

Figure 1: *The dilation 2 inorder embedding.*

along the path $\rho(e)$ in H.

Suppose we are given an embedding of G in H with dilation d and load-factor λ. It should be clear that the time to simulate one cycle of G on H can be no less than the dilation d. Furthermore, if every node in G communicates with each of its neighbors in one cycle, then as many as λ messages will pass across some edge in H in the same direction, so that the simulation must take at least λ cycles. Similarly, each message can be delayed at most λ cycles in a single queue so that if d is the dilation then $d\lambda$ cycles are sufficient to simulate one cycle of G. Summarizing, we have:

Lemma 1. *Let λ be the load-factor, and d the dilation of an embedding of G in H. If T cycles of H suffice to simulate any cycle of G using this embedding, then* $\max\{d, \lambda\} \le T \le d\lambda$.

To illustrate our definitions with an example, consider the embedding of Figure 1 of a complete binary tree within the hypercube. The nodes of the tree are numbered inorder — each node of the tree is mapped to the node in the hypercube with the corresponding address. Each edge from a node to its left child is mapped to the corresponding hypercube edge between the images of the two nodes, while the edge between a node and its right child is mapped to the path from the right child to the left child, and from the left child to the parent. The expansion, $N/(N-1)$, is the minimum possible while the dilation equals 2. The load factor also equals 2, but there are no queueing delays and two cycles of the hypercube suffice to simulate one cycle of the complete binary tree.

Since an $N-1$ node complete binary tree is never a subgraph of the N node hypercube for $N \ge 8$ [14], any embedding of a large complete binary tree in the hypercube with expansion 1 must have dilation at least 2. In this sense, the inorder embedding of Figure 1 is optimal. There is, however, a much more efficient

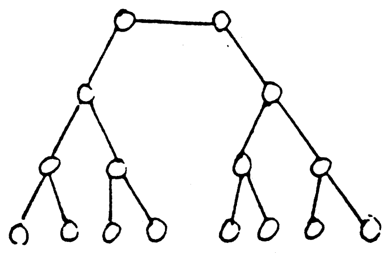

Figure 2: *A spanning tree S_4 of the hypercube.*

embedding. Bhatt and Ipsen [4] show that the N node tree $S_{\log N}$ of Figure 2 is a spanning tree of the N node hypercube. As a corollary, the $N-1$ node complete binary tree can be embedded in an N node hypercube with only one edge of dilation 2, and the with unit load-factor everywhere. In fact, this embedding is unique. Observe also that with expansion 2, the tree can be embedded with dilation 1.

Unfortunately, we do not know if arbitrarily structured binary trees can be efficiently embedded in such an elegant manner. We can use the well-known property that removing a single edge can separate a binary tree into two components each containing at least $\lfloor n/3 \rfloor$ nodes. By recursively embedding the split components within smaller hypercubes and translating one cube so that the nodes of the cut edge are adjacent in the new dimension, we obtain the following result.

Theorem 2. *Every N node binary tree can be embedded with unit-dilation in a hypercube with $O(N^{1.71})$ nodes.*

Although Theorem 2 gives a unit-dilation embedding, the expansion ($\approx N^{.71}$) is too large for the embedding to be useful in practice. In the next two sections we relax the unit-dilation requirement slightly, and show that every binary tree can be embedded with $O(1)$ dilation, expansion and load-factor.

3 The decomposition lemma

To embed an arbitrary N node binary tree T within the hypercube we proceed in two steps. In the first step T is decomposed and efficiently embedded within an N node *thistle tree*; an efficient embedding of the thistle tree within the hypercube in the second step induces an efficient embedding for T. This strategy is similar in spirit to the VLSI layout techniques of [2], with the thistle tree playing the role of the tree-of-meshes network. Combinatorial techniques developed previously for VLSI layout [2] apply in a straightforward way to the results in this section, and we will only sketch some of the proofs in this abstract. This section gives the combinatorial lemmas basic to our result; thistle trees and their embeddings are discussed in the next two sections.

Lemma 3. *Let T be any N node binary tree each of whose nodes is colored with one of k colors. Let n_i be the number of nodes of color i, $1 \leq i \leq k$, $\sum_i n_i = N$. By removing $k \log N$ or fewer edges, T can be bisected into two components of sizes $\lfloor N/2 \rfloor, \lceil N/2 \rceil$ such that, for each i, $1 \leq i \leq k$, each component has at least $\lfloor n_i/2 \rfloor$ nodes of color i.*

Lemma 4. *Every N node binary tree T can be mapped onto an N node complete binary tree C so that at most $6 \log \frac{N}{2^t} + 18$ nodes of T are mapped onto any one node of C at distance t from the root, and so that any two nodes adjacent in T are mapped to nodes at most distance 3 apart in C.*

Proof. The idea is to recursively bisect T, placing the successive sets of bisector nodes within successively lower levels of C, until T is decomposed into single nodes. For example, the nodes placed at the root of C bisect T into two subgraphs T_1 and T_2. Similarly, nodes mapped to the left child of the root bisect T_1 and nodes mapped onto the right child bisect T_2. In addition, at level i of C we map nodes of T (that have not already been mapped within levels $i-1, i-2$) that are adjacent to nodes mapped at level $i-3$ of C. This ensures that nodes adjacent in T will be mapped to nodes of C at most distance 3 apart.

To keep the number of nodes of T mapped to a level i node in C within the required bounds, we use Lemma 4 with 3 colors. The following procedure describes how this is done.

1. *Step 1.* Initialize every node of T to color A, bisect T, and place the bisector nodes at the root (level 1).
2. *Step 2.* For each subgraph created in the previous step, recolor every node adjacent to the bisector in the previous step with color 0, and place a 2-color bisector for the subgraph at the corresponding level 2 node.
3. *Step 3.* For each subgraph created in the previous step, recolor every node of color A adjacent to the bisector in the previous step with color 1, and place a 3-color bisector for the subgraph at the corresponding level 3 node.
4. *Step t.* ($\log|T| \geq t \geq 4$). For each subgraph created in the previous step, place every node of color t (mod 2) at the corresponding level t node, recolor every node of color A that is adjacent to one of color $t-1$ (mod 2) with color t (mod 2), and place a 3-color bisector for the remaining subgraph at the corresponding level t node.

If n_i is the maximum number of nodes mapped to a level i node of C, then we have $n_1 = \log N$, $n_2 = 2\log\frac{N}{2}$, $n_3 = 3\log\frac{N}{4}$, and because we use a 3-color bisector at each step, in general we have:

$$n_i \leq 3\log\frac{N}{2^i} + \frac{1}{2}n_{i-3},$$

from which the result follows. ■

The decomposition is obtained in time polynomial in the size of T because, for fixed k, a k–color bisector for a tree can be found in polynomial time.

4 Embeddings in the hypercube

The decomposition obtained in the previous section motivates the definition of thistle trees. The thistle tree T_h of height h is obtained by starting with a complete binary tree of height h and, to each node at height i (leaves are at height 1), $1 \leq i \leq h$, attaching $i-1$ additional leaves. The thistle tree T_5 is shown in Figure 3. A simple calculation shows that the thistle tree $T_{\log N}$ of height $\log N$ has $2N - \log N - 2$ nodes.

The decomposition of Lemma 4 is invoked to embed an arbitrary N node binary tree within the thistle tree. By mapping the nodes of T that are mapped to the same internal node in Lemma 4 onto the corresponding thistle in the thistle tree (with at most $O(1)$ nodes of T at a single thistle) we can obtain an embedding with expansion 1 and dilation no greater than 5. In this embedding the thistles at the top levels may have multiple nodes of T embedded within them, but there is

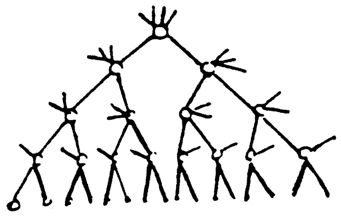

Figure 3: *The thistle tree T_5.*

a corresponding deficit at thistles at the bottom of the thistle tree. By "pushing" the excess level-by-level down the tree, we can establish the following result.

Lemma 5. *Every N node binary tree T can be mapped onto a thistle tree with expansion 1 and $O(1)$ dilation.*

It remains to embed the N node thistle tree within the hypercube in an efficient manner. To this end, consider the inorder numbering of an $N/2$ node complete binary tree, as shown in Figure 4. It is not hard to see that each node u is within distance 1 (in the $N/2$ node hypercube) of every node along the rightmost path in the left subtree of u. Embed the thistle tree so that the centre of each thistle maps to the corresponding node in the complete binary tree, and each leaf maps to a distinct node in the rightmost path of the left subtree of the central node. Notice that the length of this path always equals the number of leaves hanging off the central node in the thistle. At this point each node of the hypercube has at most two thistle tree nodes mapped onto it. Now add another $N/2$ subcube and project each leaf of a thistle onto this "shadow" tree — this gives us an embedding of the thistle tree with expansion 1 and dilation 2. Together with Lemma 5, this guarantees an embedding of arbitrary T with expansion 1 and $O(1)$ dilation.

A more careful analysis in Lemma 5 shows that nodes u, v adjacent in T are mapped to thistles at most 6 levels apart. An interesting property of the inorder numbering is that the set of nodes obtained by picking a node and its descendants at distances $1, 2, \ldots, t$ induce a $t+1$ dimensional subcube (with one node missing), so that any two nodes at most $t-1$ levels apart are within distance $t+1$ in the hypercube. Therefore, by our earlier remark, the distance between the central nodes of the thistles for u and v are distance 8 apart in the hypercube. Since every central node is within distance 2 of its leaves, the dilation of the overall embedding is at most 10. It is now straightforward to find paths in the hypercube between adjacent nodes of T so that the load factor is small. Summarizing, we have our main result of this section.

Theorem 6. *Every N node binary tree can be embedded in a hypercube with expansion 1, dilation 10 and $O(1)$ load-factor.*

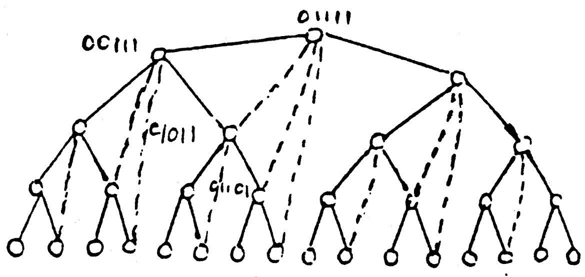

Figure 4: *Embedding the thistle tree within the complete binary tree.*

Together with Lemma 1, we have thus shown that every binary tree machine can be simulated with $O(1)$ communication overhead on a hypercube. The embedding can be computed in polynomial time because the bisection in Lemma 3, and consequently all other computations, can be computed in polynomial time.

5 An optimal universal graph

A graph H is said to be *universal* for a family of N node graphs if every graph in the family is a subgraph of H. The subgraph property is extremely strong (and attractive in applications) since it is equivalent to embeddings with unit dilation and load factor.

The problem of constructing N node universal graphs with fewest number of edges for all N node trees has received considerable attention. Following the work of [6, 7] Chung and Graham [8] constructed a universal graph for trees with $O(N \log N)$ edges. This bound is optimal, to within constant factors [5].

For binary trees, however, smaller universal graphs exist. Any N node binary tree can be embedded within an N node thistle tree with dilation 5. By connecting every pair of nodes that are at most distance 5 apart in the thitle tree, we obtain an N node graph with $O(N)$ edges that contains every N node binary tree as a spanning tree. However, the degree of the root is $O(\log N)$ so that although this universal graph is sparse, its nodes have unbounded degree.

There do however exist graphs with bounded–degree that are universal for all binary trees. This section gives the construction. First we need a few definitions.

Definition. A graph $G(V, E)$ is said to be *full* if for every $V' \subset V$, $|V'| \leq |V|/2$, the number of edges between V and V' is at least $|V'|$.

We observe in passing that there is a constant d such that for every m, there is an m node full graph with maximum degree d. In fact, any expander can be used for constructing full graphs.

The universal graph Γ on N nodes is obtained as follows. For simplicity, we will assume that $N = 2^{\alpha} - 1$. Start the construction with a complete binary tree on N nodes. Then add edges so that the nodes at level k (a constant specified later) form a full graph on k nodes. Repeat this for nodes at levels $2k, 3k, \ldots$. Call the resulting graph Γ_0.

Next, add extra edges so that the nodes at levels $k, 2k, \ldots, \log N - s$ (k divides $\log N - s$ and s is a constant specified later) collectively form a full graph. Call the resulting graph Γ_1. Finally, insert an edge between any pair of nodes within distance t of each other, where t is a constant specified later. The resulting graph, denoted Γ, is our universal graph. Observe that the maximum degree of any node in Γ is no greater than $(2d+3)^t$ which, of course, is a constant because d and t are.

Of course, the construction given is primarily of theoretical interest because of the large constants. In the following subsections we establish the main result below.

Theorem 8. *Every N node binary tree is a spanning tree of Γ.*

5.1 Flow lemmas

The proof of Theorem 8 is somewhat involved, and requires a few combinatorial lemmas concerning full graphs and trees. The intuition captured in the following lemmas may be understood as follows. Suppose that we have mapped a subset of the nodes of a tree T within a graph G, and we next wish to map a node v of T onto a node of G in such a way that it remains "close to" its neighbors that have already been embedded. If there is no place readily available, we can still find a suitable place for v by "perturbing" the existing mapping slightly to make room for v. The "flow lemmas" establish conditions under which this can be done without dilating edges significantly.

Lemma 9. *Let G be a full graph with maximum degree d, and consider any assignment of packets to nodes of G such that every node of G is assigned at least $\lceil d/2 \rceil$ packets. Then for any disjoint subsets S and T of nodes such that $|S| = |T|$, it is possible to redistribute the packets so that:*

- *every packet either stays stationary or moves to a neighbor in G,*
- *the number of packets in each node in S decreases by 1,*
- *the number of packets in each node in T increases by 1, and*
- *the number of packets in each node in $V - (T \cup S)$ remains the same.*

Proof sketch: The lemma is proved with a simple max-flow/min-cut argument. Set up a flow problem with a supersource connected to each node in S and a supersink connected to each node in T. Assign unit capacity to each edge. Because G is full, there is a $0-1$ flow with value $|S|$ between the source and sink. The flow determines a 1–1 correspondence (along with edge-disjoint paths) from the nodes in S to the nodes in T. By moving one packet forward along each edge that has unit flow we can effect a reassignment of packets that satisfies the last three conditions of the lemma.

Since every node in the flow graph (with the supersource and supersink) has degree at most $d+1$, at most $\lfloor (d+1)/2 \rfloor = \lceil d/2 \rceil$ packets will be removed from any node of G during the reassignment process. Since every node of G initially has $\lceil d/2 \rceil$ packets, no packet need ever move more than one step. Hence the reassignment also satisfies the first condition.

Lemma 10. *Let G be an m node full graph with maximum degree d, and consider any assignment of packets to nodes of G so that node v_i has a_i packets, where $a_i \geq \lceil d/2 \rceil$ for $1 \leq i \leq m$. Then, for any set of numbers $\{a'_i \mid 1 \leq i \leq m\}$ for which $a'_i \geq \lceil d/2 \rceil$ for $1 \leq i \leq m$, it is possible to redistribute the packets so that*

- *every packet is reassigned to a node which is at distance at most $\max_{1 \leq i \leq m} |a_i - a'_i|$ from its original location in G, and*
- *the number of packets assigned to v_i changes from a_i to a'_i, for all $1 \leq i \leq m$.*

Proof sketch: Apply Lemma 9 $\max_{1 \leq i \leq m} |a_i - a'_i|$ times, each time decreasing the maximum value of $\max_{1 \leq i \leq m} |a_i - a'_i|$ by one, but otherwise preserving the hypothesis of the lemma.

5.2 Decompositons revisited

To establish Theorem 8 we use a decomposition strategy different from that in section 3. The following lemma is a simple extension of the 1/3 : 2/3 separator theorem for binary trees and was observed previously in [3].

Lemma 11. *For every constant $p < 1/2$, there exists a constant q such that any m node two-colored binary forest with w nodes of color A can be partitioned into two sets by the removal of q edges so that each set has at least $\lfloor pm \rfloor$ nodes and at least $\lfloor pw \rfloor$ nodes of color A.*

We also require an additional, final lemma below.

Lemma 12. *Every N node binary tree T can be embedded within Γ_0 so that:*

- *every node in levels $0, k, 2k, \ldots, \log N - s$ of Γ_0 is assigned at least $\lceil d/2 \rceil$ and at most c_1 nodes of T, where c_1 is some constant,*
- *nodes of T are only assigned to nodes in levels $0, k, 2k, \ldots, \log N - s$ of Γ_0,*
- *nodes adjacent in T are assigned to nodes in Γ_0 separated by distance at most c_2, for some constant c_2.*

Once Lemma 12 is established, it is easy to complete the proof of Theorem 8.

Proof of Theorem 8. First obtain the embedding of Lemma 12. Next, by Lemma 10 we can use the edges of $\Gamma_1 - \Gamma_0$ to reassign the nodes of T within Γ_1 so that:

- every node in levels $0, k, 2k, \ldots, \log N - s - k$ of Γ_1 is assigned $2^k - 1$ nodes of T,
- every node in level $\log N - s$ of Γ_1 is assigned $2^s - 1$ nodes of T, and
- nodes adjacent in T are assigned to nodes in Γ_1 separated by distance at most c_3, where $c_3 \leq c_2 + 2\max(|2^s - 1 - \lceil d/2 \rceil|, |2^k - 1 - c_1|)$.

At this point, we need only require that $s \geq k$ and that $2^k - 1 \geq \lceil d/2 \rceil$ so that the conditions of Lemma 10 are satisfied. Since k, s, d, c_1, and c_2 are all constants, we know that c_3 also is constant. We now reassign nodes one more time so that the mapping from T to Γ is 1–1 and onto. This is done by arbitrarily assigning the nodes of T on levels $0, k, 2k, \ldots, \log N - s$ of Γ_1 to their immediate descendants. Once this is done, the maximum distance in Γ_1 between any two nodes adjacent in T will be at most $c_3 + 2s$, which is constant. By setting $t = c_3 + 2s$ in the to the corresponding node of Γ_0 where the enclosing subforest is currently located, making sure to map at least $\lceil d/2 \rceil$ nodes of T to each node on level k in Γ_0 (if there are not enough red nodes, then use up some of the white nodes within the same subforest to make up the total. We will show later that there are always enough nodes overall so that this is possible).

After the mapping is completed for level k, recolor red all white nodes of T that are adjacent to nodes already mapped and henceforth regard the collection of subforests assembled at a single node of Γ_0 as a single forest. Next, repeat the process used on levels $1, 2, \ldots, k$ for levels $k+1, k+2, \ldots, 2k, \ldots, \log N - s$ where s is a constant still to be specified. At every kth level we rebalance and coalesce forests as on level k, and map all red nodes of T to the corresponding nodes of Γ_0. At level $\log N - s$ all the unmapped nodes of T (both red and white) are directly mapped to the corresponding node of Γ_0. Several details remain to be ironed out; however, it should be clear that nodes adjacent in T are mapped to nodes which are at most k levels apart in Γ_0.

The analysis needed to complete the proof is tedious, but not difficult. We start by letting r_{ik} be the maximum number of red nodes in any forest after all partitioning, balancing, coalescing, mapping and recoloring is done at level ik of Γ_0. Similarly, let z_{ik} be the maximum number of nodes (both red and white) in the smallest forest at level ik.

We will prove by induction that, for $ik \leq \log N - s$,
$z_{ik} \geq 2^{-ik}N/6$,and
$r_{ik} \leq r' = 96(1+r)2^{-k}p^{-(k+\lceil \log \lceil d/2 \rceil \rceil + 1)}$

Observing that $r' \geq \lceil d/2 \rceil$, we note that both statements are trivially true for $i = 0$ and N sufficiently large. We next calculate bounds for r_{ik+k} and z_{ik+k} to proceed with the inductive step.

By Lemma 11, we know that

$$r_{ik+1} \leq (1-p)r_{ik} + 1 + q$$

and therefore, each forest at level $ik+k$ of Γ_0 has at most $(1-p)^k r_{ik} + (1+q)/p$ red nodes initially. The process of partitioning forests into subforests at level $ik+k$ cannot increase this value, but redistributing, coalescing and recoloring certainly can. To measure their effect, we need to bound the number of subforests that are located at any node following redistribution. This of course depends on the overall number of subforests, which in turn depends on the size of the smallest subforest.

The size of the smallest subforest at level ik is z_{ik}. Hence, the size of the smallest forest at level $ik+1$ is at least $pz_{ik} - 1$. Applying the argument recursively, we find that the size of the smallest subforest at level $ik+k$ (after all the subdividing construction of Γ, this will mean that T is a subgraph of Γ, thereby completing the proof. ∎

Proof of Lemma 12. We follow an approach similar to that in section 3. However, since we are allowed to place only $O(1)$ nodes of T at any one node of Γ_0, we cannot afford to bisect the tree at each step because that may require placing placing $\Theta(\log N)$ nodes of T at the root of Γ_0. Therefore, instead of bisecting the tree at each step, we separate it into proportional size components using Lemma 11, and continually balance the sizes of components as the embedding proceeds towards lower levels of Γ_0.

Start by picking any $\lceil d/2 \rceil$ nodes of T and mapping them to the root (level 0) of Γ_0. Color *red* those nodes of T that are adjacent to one or more of the nodes placed at the root of Γ_0 (all nodes are initially colored white). Next, fix p to any constant greater than 1/3 and use Lemma 11 to partition the (as yet unmapped) nodes of T into two sets, each with at least a fraction p of the total number of unmapped nodes, and each with at least a fraction p of the total number of red nodes (always rounded to the nearest integer, of course). One of the sets is distributed to the left subtree of the root of Γ_0 and the other set to the right subtree. By Lemma 11, no more than q edges connect nodes in the two sets.

No nodes of T will be assigned to the next $k-1$ levels of Γ_0, but we continue to partition T into smaller and smaller sets. In particular, we first color nodes in the "left set" of T (those unmapped nodes of T distributed to the left subtree of Γ_0 which are adjacent to nodes in the right set. We then use Lemma 11 to partition the left and right sets each into two smaller subsets, one for each grandchild of the root. Continue in this fashion, coloring nodes red as they become adjacent to nodes in the opposite set and splitting the forests (sets) into smaller forests until we have distributed a forest to each node on the kth level of Γ_0.

Although the nodes are split into roughly equal fractions $(p : 1-p)$ at each level, the sizes of forests at the kth level could vary substantially (in fact, anywhere between p^k and $(1-p)^k$). Therefore, at this stage we balance the sizes of the forests assigned to each node by redistributing forests among nodes at level k. To achieve this balance, first use Lemma 11 to partition each forest into $\lceil d/2 \rceil$ subforests (but do not distribute the subforests further down the tree). Next, partition each subforest whose size is greater than $1/p$ times the size of the smallest subforest. Observe that this does not affect the size of the smallest subforest.

We are now ready to apply Lemma 10, with each subforest represented as a packet. In particular, we use Lemma 10 to redistribute subforests on the level so that every node ends up with an equal number of subforests (to within one). We then map all the red nodes of T (i.e., those adjacent to nodes in different subforests) at this level is complete) is, for $p \leq 1/2$, at least

$$z_{ik}p^{k+\lceil\log\lceil d/2\rceil\rceil} - (1-p)^{-1} \geq p^{k+\lceil\log\lceil d/2\rceil\rceil}2^{-ik}N/6 - 2$$

For sufficiently large s (i.e., small enough i), this is at least $p^{k+\lceil\log\lceil d/2\rceil\rceil}2^{-ik}N/12$. Hence, the number of subforests at this level is no greater than $12\times 2^{ik}p^{-(k+\lceil\log\lceil d/2\rceil\rceil)}$. The maximum number of subforests located at any node after balancing is therefore no greater than

$$1 + 12 \times 2^{-k}p^{-(k+\lceil\log\lceil d/2\rceil\rceil)} \leq 24 \times 2^{-k}p^{-(k+\lceil\log\lceil d/2\rceil\rceil)}.$$

Consequently, the maximum number of red nodes in any forest after rebalancing and coalescing is at most

$$\Big((1-p)^k r_{ik} + (1+q)/p\Big)\, 24 \times 2^{-k}p^{-(k+\lceil\log\lceil d/2\rceil\rceil)}.$$

Since mapping and recoloring can increase this at most by a factor of two, we have:

$$\begin{aligned} r_{ik+k} \quad \leq \quad & 48(1-p)^k r_{ik}2^{-k}p^{-(k+\lceil\log\lceil d/2\rceil\rceil)} + \\ & 48(1+q)2^{-k}p^{-(k+\lceil\log\lceil d/2\rceil\rceil)} \end{aligned}$$

By choosing $p > 1/3$ so that $(1-p)/2p < 1$, we have that for k sufficiently large (in terms of p and d):

$$r_{ik+k} \leq \tfrac{1}{2}r_{ik} + 48(1+q)2^{-k}p^{-(k+1+\lceil\log\lceil d/2\rceil\rceil)},$$

and thus,

$$r_{ik+k} \leq 96(1+q)2^{-k}p^{-(k+1+\lceil\log\lceil d/2\rceil\rceil)} = r',$$

as claimed.

We next complete the inductive step for z_{ik+k}. Since the largest and smallest subforests differ in size by at most a factor of $1/p$, the size of the smallest forest after balancing and coalescing is at least $\frac{p}{2}(N - r'2^{ik+k})2^{-(ik+k)}$, the one–half in front accounting for the fact that every node has the same number of packets to within one. After mapping and recoloring, the size of the smalest forest is

$$z_{ik+k} \geq \tfrac{p}{2}(N - r'2^{ik+k})2^{-(ik+k)} - r'.$$

With some additional calculations it can be checked that this is at least $2^{-(ik+k)}N/6$ for $p > 1/3$ and sufficiently large (but constant) s, thereby completing the proof of the claim.

By choosing s sufficiently large, we have shown that every node at levels $0, k, \ldots, \log N - s - k$ of Γ_0 is assigned at least $\lceil d/2 \rceil$ and at most r' nodes of T. Since s is constant, every node at level $\log N - s$ of Γ_0 is aassigned between $\lceil d/2 \rceil$ and c_1 nodes, where c_1 is some constant bigger than r'. Moreover, nodes of T are only assigned to nodes in levels $0, k, \ldots, \log N - s$ of Γ_0. Hence it remains only to show that nodes adjacent in T are assigned to nodes in Γ_0 separated by distance at most c_2, for some constant c_2. We already know that c_2 is at most k plus the distance subforests are allowed to move during the rebalancing step at every kth level. By Lemma 10, this distance is at most the largest number of subforests at any node before rebalancing. By the construction, this is at most some constant determined by p, d, k and s. ∎

6 Extensions and conclusions

This paper gives the first non-trivial simulations of structures other than grids in the hypercube. The decomposition lemma (Lemma 4) for binary trees also provides optimal embeddings of binary trees within other networks. For example, we can show that every N node binary tree can be embedded within an N node complete binary tree with expansion 1 and dilation $O(\log\log N)$. This settles a conjecture of Hong, Mehlhorn and Rosenberg [11] who showed a lower bound of $\Omega(\log\log N)$ for this problem. By embedding a complete binary tree within the shuffle–exchange graph with expansion 1 and dilation 2, we obtain $O(\log\log N)$ dilation for arbitrary trees embedded within the shuffle–exchange graph. Similarly, we have recently shown that an N node complete binary tree can be embedded with constant expansion and dilation within the FFT network; once again, it follows that any N node binary tree can be embedded with constant expansion and $O(\log\log N)$ dilation within the FFT graph. We leave open the question whether these bounds are optimal to within constant factors.

All our results on embeddings within the hypercube extend to bounded degree graphs with $O(1)$ separators, and are not restricted to binary trees. While our simulations are optimal to within constant factors, there is much room for reducing the overhead in expansion and dilation further. It would be satisfying to discover simpler ways to embed binary trees in the hypercube. For example, we do not know of any binary tree that cannot be embedded in the hypercube either with expansion 1 and dilation 2 or with expansion 2 and dilation 1.

The construction of (unbounded degree) universal graphs with few edges for binary trees based on the decomposition lemma also extend to bounded–degree trees. We can also construct graphs with $O(N \log N)$ edges that are universal for bounded–degree planar graphs with N nodes. We leave open the question whether this bound is optimal to within constant factors.

An important problem concerns efficient simulations of planar graphs on the hypercube. To our knowledge, only the problem of embedding grids has been studied previously. Planar graphs arise in many scientific applications involving two-dimensional finite-element analysis. Similarly, little is known regarding lower bounds on embeddings. For example, we can prove that every N node graph with minimum bisection $\Omega(N)$ requires dilation $\Omega(\log N)$, the maximum possible. To our knowledge, no other lower bounds on embeddings in the hypercube are known.

In this paper we have only considered injective mappings of static structures. Depending upon the application, there are many interesting models. For example, if the leaves of a binary tree represent active processes and internal nodes are waiting processes, then only the leaves need be mapped to distinct nodes. In other applications, the tree may be much larger than the underlying network in which case we need to minimize dilation as well as maintain load balance. Embedding dynamically changing structures within the hypercube is important in many applications, and little is known in this area. Also interesting is the problem of *on–line* embeddings, in which the tree to be embedded grows one node at a time. We can show that any N node network for which every N node binary tree can be embedded on–line as a spanning tree must contain $\Omega(N^2)$ edges. In contrast, Friedman and Pippenger [10] show that if the size of the tree is small (a constant fraction of N) then $O(N)$ edges suffice.

Acknowledgements

Thanks to Michael Fischer and Lennart Johnsson for helpful discussions. Sandeep Bhatt was supported by NSF grants DCI 8601885 and DCR 8405478, Tom Leighton by grants AFOSR-860078, DARPA N00014-80-C0622, and an NSF Presidential Young Investigator Award with matching funds from Xerox and IBM, and Arnold Rosenberg by NSF grant DMC 8504308. The work was also supported in part by Bell Communications Research.

References

[1] J. Bentley and H.T. Kung, "A tree machine for searching problems," *Proc. Intl. Conf. Parallel Processing,* (1979).

[2] S.N. Bhatt and F.T. Leighton, "A framework for solving VLSI graph layout problems," *JCSS,* Vol. 28, No.2, (1984).

[3] S.N. Bhatt and C.E. Leiserson, "How to assemble tree machines," ***Advances in Computing Research 2,*** (F. Preparata editor) JAI press 1984.

[4] S.N. Bhatt and I. Ipsen, "Embedding trees in the hypercube," Yale University Research Report RR-443 (1985).

[5] F.R.K. Chung and R.L. Graham, "On universal graphs," *Proc. 2nd. Intl. Conf. on Combin. Math.,* (A. Gewirtz and L. Quintas, Eds.); *Ann. N.Y. Acad. Sci.* 136–140, (1979).

[6] F.R.K. Chung and R.L. Graham, "On graphs which contain all small trees," *J. Combin. Theory Ser. B* 24 14–23 (1978).

[7] F.R.K. Chung, R.L. Graham and N. Pippenger, "On graphs which contain all small trees II," *Proc. 1976 Hungarian Colloq. on Combinatorics,* 213–223, North Holland, (1978).

[8] F.R.K. Chung and R.L. Graham, "On universal graphs for spanning trees," *Proc. London Math. Soc.,* 27 203–211 (1983).

[9] W.J. Dally and C.L. Seitz, "The balanced cube: a concurrent data structure," Caltech Technical report 5174:TR:85 (1985).

[10] J. Friedman and N. Pippenger, "Expanding graphs contain all small trees," manuscript (1986).

[11] J.-W Hong, K. Mehlhorn and A.L. Rosenberg, "Cost trade-offs in graph embeddings, with applications," *J. ACM 30,* 709–728, (1983).

[12] L. Johnsson, "Basic linear algebra computations on hypercube architectures," Yale University Technical Report (1985).

[13] A.L. Rosenberg, "Issues in the study of graph embeddings," *Proc. Workshop on Graph-theoretic concepts in Computer Science,* (1980).

[14] Y. Saad and M. Schultz, "Topological properties of the hypercube," Yale University Technical Report (1985).