INTERNATIONAL COMPUTER SCIENCE INSTITUTE

1947 Center St. • Suite 600 • Berkeley, California 94704-1198 • (510) 643-9153 • FAX (510) 643-7684

Efficient Dynamic Embeddings of Binary Trees into Hypercubes

Volker Heun Ernst W. Mayr*

TR-98-023

August 1998

Abstract

In this paper, a deterministic algorithm for dynamically embedding binary trees into hypercubes is presented. Because of a known lower bound, any such algorithm must use either randomization or migration, i.e., remapping of tree vertices, to obtain an embedding of trees into hypercubes with small dilation, load, and expansion simultaneously. Using migration of previously mapped tree vertices, the presented algorithm constructs a dynamic embedding which achieves dilation of at most 9, unit load, nearly optimal expansion, and constant edge- and node-congestion. This is the first dynamic embedding that achieves these bounds simultaneously. Moreover, the embedding can be computed efficiently on the hypercube itself. The amortized time for each spawning step is bounded by $O(\log^2(L))$, if in each step at most L new leaves are spawned. From this construction, a dynamic embedding of large binary trees into hypercubes is derived which achieves dilation of at most 6 and nearly optimal load. Similarly, this embedding can be constructed with nearly optimal load ρ on the hypercube itself in amortized time $O(\rho \cdot \log^2(L/\rho))$ per spawning step, if in each step at most L new leaves are added.

*Institut für Informatik der TU München, D-80290 München, Germany

1 Introduction

Hypercubes are a very popular model for parallel computation because of their regularity and their relatively small number of interprocessor connections. Another important property of an interconnection network is its ability to simulate efficiently the communication of parallel algorithms. Thus, it is desirable to find suitable embeddings of graphs representing the communication structure of parallel algorithms into hypercubes representing the interconnection network of a parallel computer.

Embeddings of graphs with a regular structure, like rings, (multidimensional) grids, complete trees, binomial trees, pyramids, X-trees, meshes of trees and so on, have been investigated by numerous researchers, see, e.g., [7, 8, 9, 10, 11, 25, 27, 28, 29]. In general, the communication structure of a parallel algorithm can be very irregular. Embeddings of such irregular graphs, like binary trees, caterpillars, graphs with bounded treewidth, have also been studied in, e.g., [2, 5, 6, 14, 15, 16, 17, 18, 20, 23].

For arbitrary binary trees, one-to-one embeddings into their optimal hypercubes with constant dilation have been constructed in [5, 6, 17, 23]. The embedding given in [17] yields dilation 8 and constant node-congestion, which is the best known bound on the dilation. Furthermore, this embedding can be efficiently computed on the hypercube itself. In [14], Havel has conjectured that every binary tree has a one-to-one embedding into its optimal hypercube with dilation at most 2. This conjecture is still open. In terms of lower bounds, a simple parity argument shows that the complete binary tree of size 2^d-1 cannot be a subgraph of the d-dimensional hypercube, see [7, 25, 29].

All these embeddings are constructed as *static* embeddings, which means that the whole information about the structure of the guest graph has to be known in advance. Since the guest graph represents the communication structure of a parallel algorithm, the guest graph may vary during the execution of the algorithm. Thus, it is important to investigate *dynamic* embeddings of graphs. Static embeddings are usually much easier to construct than dynamic embeddings. Moreover, it might be impossible to construct dynamic embeddings deterministically with high quality. For arbitrary binary trees, it has been proved that dynamic embeddings cannot be constructed with high quality if neither randomization nor migration, i.e., remapping of tree vertices, is allowed [21, 22].

For embedding complete binary trees dynamically into optimal hypercubes, optimal deterministic algorithms have been presented in [12, 19]. These embeddings achieve dilation 2, unit load, and unit congestion, while the running time is constant for each new level of leaves. A first challenge of dynamically embedding arbitrary binary trees into hypercubes was the work in [3, 4]. It presents a randomized algorithm for embedding binary trees into hypercubes with dilation $O(\text{loglog}(n))$ and, with

high probability, constant load. This was improved in [21, 22], where a randomized algorithm for embedding binary trees into hypercubes with dilation 8 and, with high probability, constant load was explored. The edge-congestion of the embedding is constant, whereas its node-congestion is $\Theta(\log(n))$.

In this paper, we will develop a deterministic algorithm using migration for embedding binary trees into hypercubes. It will be shown that a binary tree can be dynamically embedded into a hypercube with dilation of at most 9, unit load, nearly optimal expansion, and constant edge- and node-congestion. Moreover, the embedding can be computed on the hypercube itself. The amortized time for each new vertex is constant if in each step at most one new leaf is spawned. If in each step a group of at most L new leaves gets spawned, the amortized cost for each new group of leaves is bounded by $O(\log^2(L))$.

Furthermore, this construction can be extended for embedding large binary trees. We prove that a binary tree of size M can be dynamically embedded into a hypercube of size N with dilation of at most 6 and nearly optimal load. This embedding can be computed on the hypercube itself in amortized time $O(M/N \cdot \log^2(L \cdot N/M))$ per spawning step, if the binary tree grows by at most L leaves per step.

Remarkably, the quality of our dynamic embedding improves all previously known results and is the first algorithm which achieves nearly optimal values for all measures of the quality of an embedding. In particular, compared to [21, 22], our extended construction yields a dynamic embedding with dilation of at most 6, constant node-congestion, and a small nearly optimal load.

The remainder of this paper is organized as follows. First, we recall some basic definitions and notations which we will use later. In the third section, we review a lower bound on dynamic embeddings of binary trees into hypercubes. We recall the basic results on statically embedding binary trees into hypercubes in the fourth section and introduce our main tool for dynamic embeddings, namely the (h, o, τ)-tree. In the fifth section, we present the deterministic algorithm for dynamic embedding and analyze its quality as well as its amortized time complexity. We then discuss an extension of this technique for embedding large binary trees with nearly optimal load in the sixth section. Finally, we give some concluding remarks.

2 Preliminaries

An *embedding* of a graph $G=(V_G, E_G)$, called *guest graph*, into a graph $H=(V_H, E_H)$, called *host graph*, is a mapping $\varphi: G \to H$ consisting of two mappings $\varphi_V: V_G \to V_H$ and $\varphi_E: E_G \to \mathcal{P}(H)$. Here, $\mathcal{P}(H)$ denotes the set of paths in the graph H. The mapping φ_E maps each edge $\{v, w\} \in E_G$ to a path $p \in \mathcal{P}(H)$ connecting $\varphi_V(v)$ and $\varphi_V(w)$. We call an embedding *one-to-one* if the mapping φ_V is 1-1.

The *dilation* of an edge $e \in E_G$ under an embedding φ is the length of the path $\varphi_E(e)$. Here, the length of a path p is the number of its edges. The *dilation of an embedding* φ is the maximal dilation of an edge in G. The number of vertices of a guest graph which are mapped onto a vertex v in the host graph, is called the *load* of the vertex v. The *load of an embedding* φ is the maximal load of a vertex in the host graph. The ratio $|V_H|/|V_G|$ is called the *expansion* of the embedding φ. The *congestion* of an edge $e' \in E_H$ is the number of paths in $\{\varphi_E(e) \mid e \in E_G\}$ that contain e'. The *edge-congestion* is the maximal congestion over all edges in H. The *congestion* of a vertex $v \in V_H$ is the number of paths in $\{\varphi_E(e) \mid e \in E_G\}$ containing v. Again, the *node-congestion* is the maximal congestion over all vertices in H. In the following, we initially restrict our attention to finding a suitable mapping φ_V, and we will use shortest paths in the hypercube for the mapping φ_E. Nevertheless, it is still important to decide which paths we choose, since we are interested in obtaining an embedding with small node-congestion.

A *hypercube of dimension* d is a graph with 2^d vertices, labeled 1-1 with the strings in $\{0,1\}^d$. Two vertices are connected iff their labels differ in exactly one position. For convenience, we will often identify a hypercube vertex with its label. The smallest hypercube into which a given graph $G=(V,E)$ can be embedded with unit load is called its *optimal* hypercube. Thus, the dimension of the optimal hypercube is $\lceil \log(|V|) \rceil$. The next larger hypercube of dimension $\lceil \log(|V|) \rceil + 1$ is called its *next to optimal hypercube*. Hence, a one-to-one embedding of a graph G into its optimal or next to optimal hypercube has expansion less than two or four, respectively.

As usual, a *tree* is a connected acyclic graph with one distinguished vertex, called the *root*. A vertex of degree 1 is called a *leaf* if it is not the root. A vertex is called an *internal* vertex if it is not a leaf. Given a vertex v in a tree, another tree vertex w is called a *successor* of v if v lies on the simple path from the root to w. We call v also an *ancestor* of w. A vertex w is called a *child* of a vertex v if w is a successor of v and v and w are adjacent. The vertex v is also called a *parent* of w. The *level* of a tree vertex v is the number of vertices on the simple path from the root to v. For instance, the level of the root is 1. The *height* of a tree T is the maximum level of a vertex in T. A *subtree rooted at a vertex* v is the induced subgraph of all successors of v in the tree. In the sequel, we mean by a subtree always a subtree rooted at some vertex of the tree. Let v and w be two tree vertices, we call u the *lowest common ancestor* of v and w if u is the root of a smallest subtree containing v and w.

3 Lower Bounds

In this section, we briefly review that any deterministic algorithm for dynamically embedding binary trees cannot achieve constant load, dilation, and expansion without

migration. This was first proved in [21, 22] for embeddings with load greater than one:

Theorem 1 ([21, 22]) *Any deterministic algorithm for dynamically embedding binary trees of size M into a hypercube of size $N \leq M$ that achieves load $c \cdot M/N$ and does not use migration must have average dilation $\Omega((\log(N))^{\frac{1}{2}}/c^2)$.*

In fact, the proof was given for caterpillars with maximal degree 3. Using the same technique as in [21, 22], the following theorem concerning one-to-one embeddings into hypercubes can be proved.

Theorem 2 *Any deterministic algorithm for dynamically embedding binary trees of size M into a hypercube of size $N \geq M$ that achieves unit load and does not use migration must have average dilation $\Omega((\log(N))^{\frac{1}{2}}/e^2)$, where $e=\frac{N}{M}$ is the expansion of the embedding.*

Proof: We call the number of 1's in the label of a hypercube location v the *weight* of the vertex v. We partition the hypercube locations into $\ell := \lceil 6e \rceil$ sets $\mathcal{V}_i$ of size $\lceil \frac{N}{\ell} \rceil$ such that each $\mathcal{V}_i$ corresponds to some contiguous range of weights. Since the number of hypercube locations of a fixed weight is bounded by $O(N/(\log(N))^{\frac{1}{2}})$, each $\mathcal{V}_i$ contains at least $\Omega((\log(N))^{\frac{1}{2}}/e)$ vertices of pairwise different weights. Given two vertices $v \in \mathcal{V}_i$ and $w \in \mathcal{V}_{i+j}$ for $i \in [1{:}\ell - j]$ and $j \geq 2$, we can conclude that v and w differ in at least $\Omega((\log(N))^{\frac{1}{2}}/e)$ positions.

First, we grow a path of length $\frac{M-6}{2}$, called the backbone, in the hypercube. There exists an $i \in [1{:}\ell]$ such that $\mathcal{V}_i$ contains at least $\frac{M-6}{2\ell} \geq \frac{M-6}{2(6e+1)}$ hypercube locations which are images of the backbone. From each vertex in the backbone whose image is contained in $\mathcal{V}_i$, we grow paths (the legs) until the image of the end of such a path is no longer contained in $\mathcal{V}_{i-1} \cup \mathcal{V}_i \cup \mathcal{V}_{i+1}$. The number of hypercube locations which are images of the vertices of the legs is at most $3\lceil \frac{N}{\ell} \rceil \leq \frac{3N}{6e} + 3 = \frac{M+6}{2}$. Hence, the size of the caterpillar is at most M. As explained above, the Hamming distance of the endpoints of these legs is at least $\Omega((\log(N))^{\frac{1}{2}}/e)$. Thus, the average dilation of the embedding is at least $\frac{1}{M} \cdot \frac{M-6}{12e+2} \cdot \Omega((\log(N))^{\frac{1}{2}}/e) = \Omega((\log(N))^{\frac{1}{2}}/e^2)$. ■

4 Review of the Static Embedding

In this section, we review the main results stated in [17] which are needed for our construction of a dynamic embedding. The details of the embedding are slightly different from [17] in order to obtain a presentation more suitable for our purposes.

4.1 Definition of a (h, o, τ)-Tree

To construct our embedding, we use the data structure of a (h, o, τ)-tree. The (h, o, τ)-*tree* (pronounced hot tree) is a complete quadtree of height h with integer node weights, also called the *capacities* of the nodes. The capacities depend on the level of a vertex, and on the parameters o and τ, with $o \in \mathbb{N}$ and $\tau \in \{0, 1\}$. We also distinguish between a *full* and a *partial* (h, o, τ)-tree, which differ only in the capacity of the root. A node at level ℓ of a full or partial (h, o, τ)-tree has a capacity of $2^o(6(h-\ell+1)-5+3\tau)+\delta_{1,\ell}2^o(2h+1+\tau)$ or $2^o(6(h-\ell+1)-5+3\tau)$, respectively, Here, $\delta_{i,j}$ denotes the Kronecker symbol. In the following, we call vertices of a (h, o, τ)-tree *nodes*, and we denote the capacity of a node at level ℓ by $c(\ell)$. Unless stated otherwise, we mean by a (h, o, τ)-tree a full (h, o, τ)-tree. Note that a subtree of a (h, o, τ)-tree is itself a partial (h, o, τ)-tree.

Lemma 3 *Let T be a subtree of a (h, o, τ)-tree rooted at a node at level ℓ, then T itself is a partial $(h-\ell+1, o, \tau)$-tree.*

In the following lemma, we determine the overall capacity of a (h, o, τ)-tree that is the sum of all capacities.

Lemma 4 *The overall capacity of a full and a partial (h, o, τ)-tree are $2^{2h+o+\tau}$ and $2^{2h+o+\tau}-2^o(2h+1+\tau)$, respectively.*

Proof: We prove only the claim for a full (h, o, τ)-tree, since the claim for partial (h, o, τ)-tree follows then immediately. By definition of the capacities we get:

$$\begin{aligned}
\sum_{\ell=1}^{h} 4^{\ell-1} c(\ell) &= \sum_{\ell=1}^{h} 4^{(\ell-1)}[2^o(6(h-\ell+1)-5+3\tau)] + 2^o(2h+1+\tau) \\
&= 2^o\left[(6h-5+3\tau)\sum_{\ell=0}^{h-1} 4^\ell - 6\sum_{\ell=0}^{h-1} \ell 4^\ell + 2h+1+\tau\right] \\
&= 2^o\left[(6h-5+3\tau)\frac{4^h-1}{4-1} - 6\frac{(h-1)4^{h+1} - h4^h + 4}{(4-1)^2} + 2h+1+\tau\right] \\
&= 2^o\left[\frac{3(1+\tau)4^h - 6h - 3 - 3\tau}{3} + 2h+1+\tau\right] \\
&= 2^o 2^{2h}(1+\tau) \quad = \quad 2^{2h+o+\tau}
\end{aligned}$$

■

4.2 Embeddings Using the (h, o, τ)-Tree

We now describe a mapping of a (h, o, τ)-tree into its optimal hypercube such that each node of the (h, o, τ)-tree occupies as many vertices of the hypercube as given by its capacity. Each node in a (h, o, τ)-tree can be represented by a string in $(\{0,1\}^2)^*$ as follows. The empty string ε represents the root of the (h, o, τ)-tree. If α represents a node v of the (h, o, τ)-tree, then the strings $\alpha 00$, $\alpha 01$, $\alpha 10$, and $\alpha 11$ represent the four children of v from left to right. Note that the string representing a node at level ℓ has length $2(\ell-1)$. For convenience, we will often identify a (h, o, τ)-tree node with its representation. We define the following sets of hypercube locations, where α represents some arbitrary node in a (h, o, τ)-tree:

$$\begin{aligned} S_\alpha &:= \left\{\alpha\beta\gamma\delta \in \{0,1\}^{2h+o+\tau} : \beta{\in}\{1, 01, 11\} \wedge \gamma{\in}0^*10^* \wedge \delta{\in}\{0,1\}^o\right\} \\ R &:= \left\{\gamma\delta \in \{0,1\}^{2h+o+\tau} : \gamma{\in}(0^*10^*{+}0^*) \wedge \delta{\in}\{0,1\}^o\right\} \end{aligned}$$

We also define the set $S{:=}\bigcup_\alpha S_\alpha$. The vertices of the binary tree mapped to the node of a full (h, o, τ)-tree represented by α will finally be mapped to hypercube locations in the set $L_\alpha{:=}S_\alpha$ if $\alpha{\neq}\varepsilon$, and $L_\varepsilon{:=}S_\varepsilon{\cup}R$ otherwise. In case of a partial (h, o, τ)-tree, we redefine $L_\varepsilon{:=}S_\varepsilon$. It can easily be verified that the capacity of a node in the (h, o, τ)-tree is equal to the cardinality of the set of vertices in the hypercube to which it is mapped.

Lemma 5 *Let α be a representation of a (h, o, τ)-tree node at level ℓ, then we have $|S_\alpha|{=}2^o(6(h{-}\ell{+}1){-}5{+}3\tau)$. Furthermore, $|R|{=}2^o(2h{+}1{+}\tau)$ and $|L_\alpha|{=}c(\ell)$.*

It can easily be verified that for every $\alpha{\neq}\alpha'{\in}(\{0,1\}^2)^*$, $S_\alpha{\cap}S_{\alpha'}{=}\emptyset$ and $S_\alpha{\cap}R{=}\emptyset$, and hence $L_\alpha{\cap}L_{\alpha'}{=}\emptyset$ for every $\alpha{\neq}\alpha'{\in}(\{0,1\}^2)^*$. Hence, for each string $s{\in}S{\cup}R$, there is a unique decomposition $s{=}\alpha\beta\gamma\delta$ as used in the definition of S_α and R. Given a hypercube location u, we call the node α of the (h, o, τ)-tree such that $u{\in}L_\alpha$ its *corresponding* node.

A simple observation from the definition of the sets L_α shows that the hypercube vertices associated with a subtree of a (h, o, τ)-tree are concentrated in a small subcube of the original h-dimensional hypercube. Note the correspondence to Lemma 3.

Lemma 6 *Let T be a subtree of a (h, o, τ)-tree of height h' and A_T the set of representations of nodes belonging to T, then $\bigcup_{\alpha\in A_T} L_\alpha$ is contained in a $2h'{+}o{+}\tau$-dimensional subcube.*

The following lemma is fundamental to prove the claimed bounds on the dilation of our embeddings.

Lemma 7 *Let α_1 and α_2 represent two nodes in a (h, o, τ)-tree such that their lowest common ancestor represented by α is at distance $\leq\Delta$ from both nodes. Let $v\in L_{\alpha_1}$ and $w\in L_{\alpha_2}$, then v and w differ in at most $2\Delta+o+4$ positions.*

Proof: We first consider the case where both v and w belong to the set S. The diagram in Figure 1 shows the unique decomposition of v and w. In this picture, α represents the lowest common ancestor of the (h, o, τ)-tree nodes represented by α_1 and α_2. Therefore, $\alpha_1=\alpha\alpha'$ and $\alpha_2 = \alpha\alpha''$. Without loss of generality, we assume $|\alpha'|\leq|\alpha''|$. Since the lowest common ancestor of the (h, o, τ)-tree nodes represented by α_1 and α_2

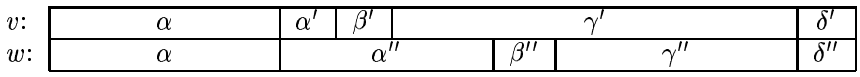

Figure 1: A Pair of Adjacent Tree Vertices Mapped to v and w

is at distance of at most Δ from both nodes, we get $|\alpha'|\leq|\alpha''|\leq2\Delta$. The definition of the sets L_α implies that $|\beta'|, |\beta''|\leq2$, $|\delta'|=|\delta''|=o$, and that γ' and γ'' contain exactly one 1 each. Hence, the labels of v and w differ in at most $2\Delta+2+1+1+o=2\Delta+o+4$ positions.

We now consider the case that $v\in R$ and that $w\in S$, i.e., $v=\gamma'\delta'$ and $w=\alpha''\beta''\gamma''\delta''$. Note that in this case necessarily $\alpha=\varepsilon$. Since the distance between corresponding (h, o, τ)-tree nodes of v and w is Δ, we have $|\alpha''|\leq2\Delta$. Hence there are at most $2\Delta+2+1=2\Delta+3$ 1's in the first $2h+\tau$ positions in w. By definition of the set R, v has at most one 1 in the first $2h+\tau$ positions. Thus, v and w differ in at most $2\Delta+3+1+o=2\Delta+o+4$ positions.

Finally, if both vertices belong to the set R, then the definition of R implies immediately that their labels differ in at most $o+2$ positions. ■

4.3 Outline of the Static Embedding

Our embedding of binary trees into hypercubes is achieved in two steps. First, we embed the binary tree into a (h, o, τ)-tree. Then, we use the mapping presented in the previous subsection to complete the embedding. To obtain a small dilation, adjacent vertices of the binary tree will be mapped to nodes which are close in the (h, o, τ)-tree.

Our goal is to obtain an embedding of the binary tree into a (h, o, τ)-tree such that adjacent tree vertices are mapped to nodes in the (h, o, τ)-tree with distance of at most one from their lowest common ancestor of the (h, o, τ)-tree. It can be shown that this goal can be reached, except for tree vertices that get mapped to (h, o, τ)-tree nodes close to the leaves, where distance of at most two can be obtained. Our

method leads to an embedding of the binary tree into the hypercube with dilation of at most $8+o$. The number o can be chosen as 3. Using local modifications of the mapping, we can then reduce the dilation to 9. The dilation given here is greater than in [17], where we achieve dilation 8, but it yields a simpler description for our dynamic embedding.

Now, we are ready to describe the embedding of an arbitrary binary tree into a (h, o, τ)-tree. The embedding proceeds in $h{=}\Theta(\log(n))$ stages. Each stage is associated with a level of the (h, o, τ)-tree. At the first stage, we fill up the root of the (h, o, τ)-tree with tree vertices and mark all unmapped neighbors of the mapped tree vertices. Then, we decompose the binary tree into four parts by removing some tree edges using Lemma 8 stated below and associate the four parts with the four children of the root.

In the subsequent stages, we map the marked vertices to the corresponding node in the (h, o, τ)-tree. Also all vertices which are incident to edges removed during the previous decomposition into four parts are mapped to their corresponding nodes. Then, we fill up each node of the considered level with vertices from the binary tree in such a way that each mapped tree vertex has at most two unmapped neighbors. Again, we mark all unmapped neighbors of mapped tree vertices and decompose the forest into four 'balanced' forests as above. This procedure will be iterated in parallel until the leaves of the (h, o, τ)-tree are reached. The proof of the bound on the dilation of the static embedding is based on the following lemma, which claims that the required decomposition of binary trees can be achieved by removing only a 'few' tree edges.

Lemma 8 ([17]) *Let $F{=}(V, E)$ be a forest of binary trees containing marked vertices and let $V'{\subseteq}V$ be the set of marked vertices. There exists a set $E'{\subseteq}E$ of size at most $6\lfloor\log(|V|)\rfloor{-}2$ and a partition of $F'{=}(V, E{\setminus}E')$ into four forests $F_k{=}(V_k, E_k)$, for $k{\in}[1{:}4]$, such that $\bigcup_{k=1}^{4} V_k{=}V$, $V_i{\cap}V_j{=}\emptyset$ for $i{\neq}j$, $\lfloor|V|/4\rfloor{\leq}|V_k|{\leq}\lceil|V|/4\rceil$, and $E{\setminus}E'{=}\bigcup_{k=1}^{4} E_k$. Furthermore, for each $k{\in}[1{:}4]$, the number of marked vertices and vertices incident to edges in E' in each F_k is at most $\lceil|V'|/4\rceil{+}3\lfloor\log(|V|)\rfloor$.*

Using this lemma, a sophisticated analysis shows that the load of each (h, o, τ)-tree node is bounded by its capacity. For more details on the bound of the dilation of the static embedding, we refer to [17]. However, based on Lemma 8, we will prove later that our dynamic embedding achieves dilation of at most 9.

4.4 Complexity of the Static Embedding

As shown in [17], the partition of a forest of binary trees as stated in Lemma 8 can be computed efficiently on the hypercube. Based on this efficient construction of the partition, efficient algorithms for embedding binary trees has been developed in [17].

Theorem 9 ([17]) *Let $F=(V,E)$ be a forest of binary trees of size n. There exists a one-to-one embedding of F into its optimal hypercube with dilation at most 9 and constant node-congestion. This embedding can be computed on the optimal hypercube in time $O(\log^2(n)\,\mathrm{logloglog}(n)\log^*(n))$ provided that the forest is stored one vertex per processor.*

Provided that the Euler contour path for the forest of binary trees is already constructed and stored in an appropriate way, the running time of this algorithm can be improved. As usual, the *Euler contour path* of a binary tree is a linear list defined as follows. For each vertex v, we introduce three vertices $\textsc{Left}(v)$, $\textsc{Right}(v)$, and $\textsc{Lower}(v)$, called list vertices. We define recursively how to link these list vertices together. Consider the subtree rooted at vertex v. First, if v has a left child, link $\textsc{Left}(v)$ with the beginning of the Euler contour path of its left subtree and the end of this Euler contour path with the list vertex $\textsc{Lower}(v)$, and link $\textsc{Left}(v)$ with $\textsc{Lower}(v)$ otherwise. Next, if v has a right child, link $\textsc{Lower}(v)$ with the beginning of the Euler contour path of its right subtree and link the end of this Euler contour path with $\textsc{Right}(v)$, and link $\textsc{Lower}(v)$ with $\textsc{Right}(v)$ otherwise. This definition of a Euler contour path can be extended to a forest of binary trees by combining the Euler contour paths of all binary trees to a single list. A linear list of length M is *stored linearly* in a hypercube of size N, if, for $i\in[1{:}M]$, the i-th vertex is stored in the hypercube vertex with label $\lfloor (i-1)/\lceil \frac{M}{N} \rceil \rfloor$. Using these notations, we are able to state the following theorem proved in [17].

Theorem 10 ([17]) *Let $F=(V,E)$ be a forest of binary trees of size n. There exists a one-to-one embedding of F into its optimal hypercube with dilation at most 9 and constant node-congestion. If F is stored one vertex per hypercube processor and the Euler contour path of F is stored linearly in the hypercube, this embedding can be computed in time $O(\log^2(n))$ on the optimal hypercube.*

5 Dynamic Embedding

In this section, we present our dynamic embedding for binary trees based on the static embedding reviewed in the previous section. For the dynamic embedding, we use a slightly larger hypercube than necessary. Although it is possible to embed a binary tree dynamically into its optimal hypercube, the slack of the hypercube gives us the opportunity to compute it efficiently.

5.1 Outline of the Dynamic Embedding

During the dynamic embedding, we distinguish between spawning and migration steps which alternate. In a spawning step, each tree vertex with fewer than two children

can spawn a new child or not. In the subsequent migration step, the embedding will be recomputed in some subtrees of the (h, o, τ)-tree, because after the spawning step the load of some (h, o, τ)-tree nodes can exceed its capacity. This recomputation will ensure that the embedding again achieves unit load while maintaining a dilation of at most 9.

After a spawning step, the embedded tree is remapped such that the load of each (h, o, τ)-tree node is less than or equal its capacity. We will not recompute the embedding of the whole binary tree, because this would be too expensive. Instead, we only recompute the embedding into sufficiently small subtrees of the (h, o, τ)-tree. A fundamental observation is that we can find a one-to-one mapping of hypercube locations corresponding to internal nodes to hypercube locations corresponding to leaves of the subtree. Moreover, the Hamming distance of such a pair of hypercube locations is at most 3. In the first phase of the remapping, the newly created leaves at internal (h, o, τ)-tree nodes are mapped to the leaves given by the mapping above. Now all newly created leaves are mapped to leaves of the (h, o, τ)-tree implying that the load of each internal (h, o, τ)-tree node is less than or equal to its capacity. In the second phase, we first determine the subtrees of the (h, o, τ)-tree for which the embedding has to be recomputed. Then we recompute the embedding for all these subtrees in parallel using our algorithm for static embeddings given in the previous section.

During a recomputation of the embedding into a subtree T of the (h, o, τ)-tree, there exist tree vertices which are mapped to T whose parents are not mapped to T or the parent of the root of T. It is necessary to ensure that such tree vertices are mapped close to their parents in the hypercube. To maintain this condition, it is sufficient, as we shall see later, to map these tree vertices to the children or grandchildren of the root of T.

5.2 Mapping New Leaves to Leaves of the (h, o, τ)-Tree

We now will show how to assign each hypercube location whose corresponding node is an internal vertex a hypercube location whose corresponding node is a leaf of the (h, o, τ)-tree. Without loss of generality, we consider in the following a $(h, 0, \tau)$-tree, since the extension to a (h, o, τ)-tree is straight forward. Let v be a hypercube location which corresponds to an internal vertex of the $(h, 0, 1)$-tree. We denote by $\lambda(v)$ the assigned hypercube location corresponding to a leaf of the $(h, 0, 1)$-tree. The main idea is to assign to a hypercube vertex v a hypercube vertex $\lambda(v)$ corresponding to a leaf of the $(h, 0, \tau)$-tree such that v and $\lambda(v)$ differ in at most three positions in the last four positions.

We first consider a $(h, 0, 1)$-tree. It has by definition 4^{h-1} leaves, and each leaf has a capacity of 4. Thus, the overall capacity of all leaves is 2^{2h} which is exactly one

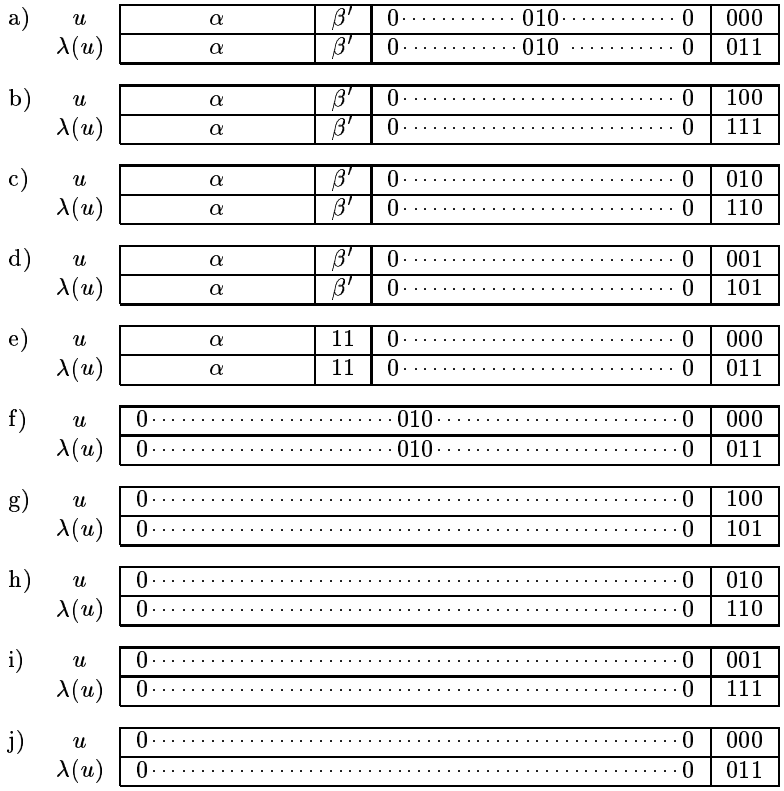

Figure 2: The Mapping λ in a $(h, 0, 1)$-Tree

half of the overall capacity of the (h, o, τ)-tree. We present a mapping such that λ is one-to-one. The mapping λ is given in Figure 2 on this page, where $\alpha \in (\{0,1\}^2)^*$ and $\beta' \in \{01, 10, 11\}$. The mapping λ for hypercube locations corresponding to a $(h, 0, 1)$-tree node represented by α is given in rows a) through e). Note that row e) corresponds to the case where β=1 and $\gamma \in 10^*$. In rows f) through j), the mapping λ for hypercube locations in the additional set R belonging to the root of the $(h, 0, 1)$-tree is given. Note that by definition a hypercube label corresponds to a leaf of a $(h, 0, 1)$-tree iff it has at least two 1's in its last three positions.

We now consider a $(h, 0, 0)$-tree. As follows from the definition, it has 4^{h-1} leaves and each leaf has a capacity of 1, which altogether is 2^{2h-2}. Since this is not one half of the capacity of the $(h, 0, 0)$-tree, we will also use some of the hypercube locations corresponding to the parents of the leaves as images of our mapping λ. Of course, such hypercube locations corresponding to parents of leaves will be removed from the domain of λ. Since $2^{2h-1} = 4^{h-1} + 4 \cdot 4^{h-2}$, we need from each parent of a leaf four additional (of $c(h-1)=7$) corresponding hypercube locations. The hypercube locations corresponding to parents of leaves with 01 in the final two positions of their labels belong to the domain of the mapping λ, the other hypercube locations ending with

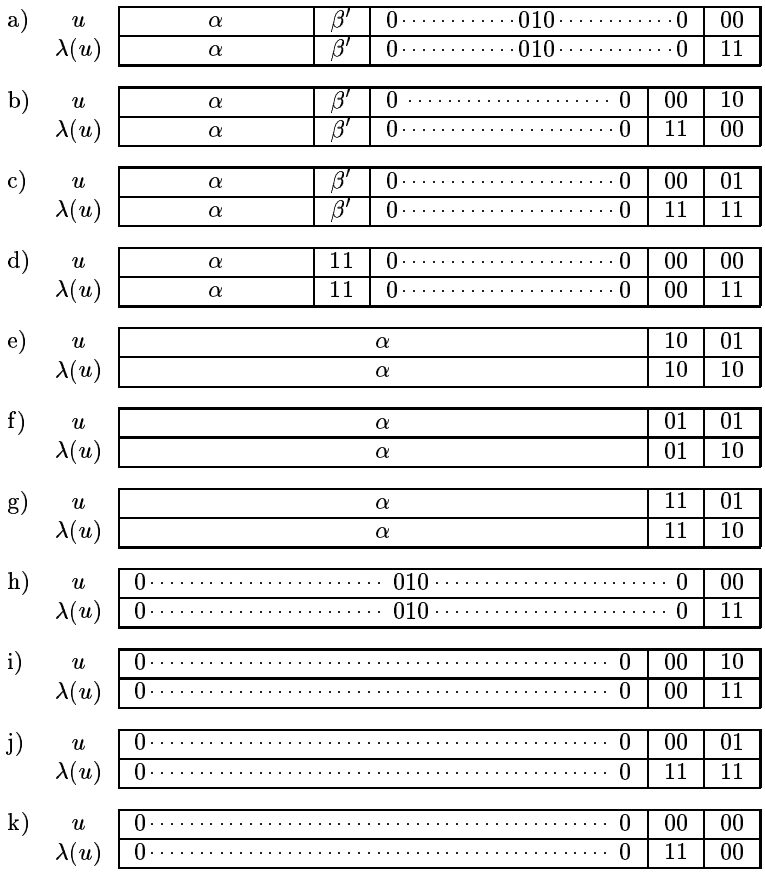

Figure 3: The Mapping λ in a $(h, 0, 0)$-Tree

10 and 00 will be used in the image of λ.

The mapping λ is illustrated in Figure 3 on this page, where $\alpha \in (\{0,1\}^2)^*$ and $\beta' \in \{01, 10, 11\}$. First we consider hypercube locations corresponding to a node at level $\ell \geq h-2$ of a $(h, 0, 0)$-tree. For a hypercube location whose corresponding node is represented by α, the mapping λ is illustrated in rows a) through d). Note that row d) corresponds to the case where $\beta = 1$ and $\gamma \in 10^*$. Next we consider hypercube locations corresponding to parents of leaves. Recall that for each node only three of seven hypercube locations remain in the domain of λ. The images of such hypercube locations also correspond to parents of leaves. The mapping λ for these parents of leaves is given in row e) through g). In rows h) through k) the mapping λ for hypercube locations in the additional set R belonging to the root of the $(h, 0, 0)$-tree is given. Note that by definition a hypercube label corresponds to a leaf of a $(h, 0, 0)$-tree iff it ends with two 1's and that a hypercube label corresponds to a node at one of the two highest numbered levels of a $(h, 0, 0)$-tree iff it has at least two 1's in the

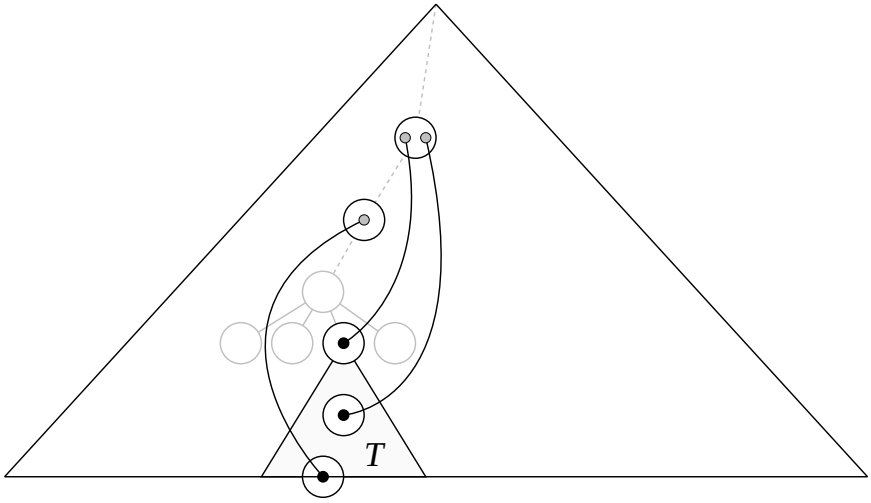

Figure 4: Sprouts in a Subtree T of a (h, o, τ)-Tree

last four positions.

An important property of the mapping λ is that the corresponding (h, o, τ)-tree node of the hypercube vertex u is always an ancestor of the corresponding (h, o, τ)-tree node of the hypercube vertex $\lambda(u)$.

5.3 Embedding Sprouts During a Reconstruction

As mentioned earlier, during a recomputation of the embedding into a subtree of the (h, o, τ)-tree, there exist tree vertices mapped to the subtree whose parents are not. Consider a fixed subtree T of the (h, o, τ)-tree. A tree vertex mapped to T is called a *sprout in* T, if its parent is not mapped to T or to the parent of the root of T. Whenever it is clear which subtree is involved, we call such a tree vertex simply a sprout. All other tree vertices which are not sprouts are called *wooden vertices in* T. The locations of sprouts in a subtree T of a (h, o, τ)-tree are illustrated in Figure 4 on page 13. Here the sprouts are drawn black.

Consider a subtree T rooted at a node at level ℓ represented by α. This implies that the height of T is $h-\ell+1$. We will count the sprouts mapped to T, which will be denoted by $t(\ell)$. By Lemma 3, T is itself a partial $(h-\ell+1, o, \tau)$-tree. By Lemma 6, the corresponding hypercube locations are contained in an $(h-\ell+1)$-dimensional subcube. Also recall that by construction of the mapping λ, for each hypercube vertex u, its corresponding (h, o, τ)-tree node is an ancestor of the corresponding (h, o, τ)-tree leaf $\lambda(u)$. Hence, parents of sprouts must be mapped to a hypercube location contained in the set

$$R_\alpha := \left\{ \alpha\gamma\delta \in \{0,1\}^{2h+o+\tau} : \gamma \in 0^*10^*+0^* \wedge \delta \in \{0,1\}^o \right\}.$$

This set R_α plays for the $(h-\ell+1, o, \tau)$-tree rooted at a node represented by α the same role as $R=R_\varepsilon$ for the whole (h, o, τ)-tree. By construction, the cardinality of R_α is thus $2^o(2(h-\ell+1)+1+\tau)$. Since each tree vertex can spawn at most two leaves, the number of sprouts mapped to a subtree T is at most $2^{o+1}(2(h-\ell+1)+1+\tau)$.

It remains to investigate what happens when the mapping in T is recomputed. There are two cases. We first assume that the parent of a sprout is not involved in the remapping. In this case, the sprouts could be remapped but their parents (which are mapped to a node outside the subtree) remain at their hypercube locations. It follows that the tree vertices mapped to the set R_α cannot spawn new sprouts, since each tree vertex has at most two children. Therefore, the number of sprouts remain unchanged. Otherwise, the parent of a sprout is also remapped. This means that we recompute a new embedding corresponding to a subtree of the (h, o, τ)-tree also containing the image of the parent of the sprout. But this implies that the sprout becomes a wooden vertex of T. Altogether, we have proved the following lemma.

Lemma 11 *Let T be a subtree of a (h, o, τ)-tree rooted at a node at level ℓ, then the number of sprouts in T is bounded by $2^{o+1}(2(h-\ell-1)+1+\tau)$.*

Consider a subtree T of the (h, o, τ)-tree and a sprout v in T and let u be the parent of v. In what follows, we determine the dilation of the edge connecting a sprout and its parent provided that the sprout is either mapped to a child or a grandchild of the root of T. In Figure 5, the labels of the sprout v and its parent u are shown. Here, α represents the root of the subtree T. Row a) shows the label if v is mapped to a child of the root of T and row b) shows the label if v is mapped to a grandchild. In this

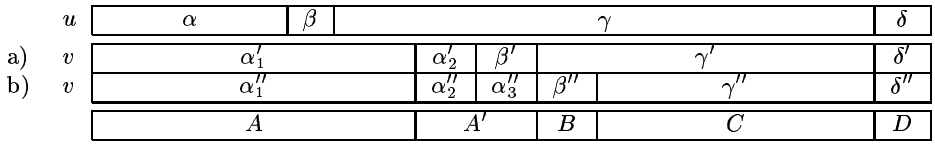

Figure 5: Possible Hypercube Locations of a Sprout v and Its Parent u

figure, $\alpha\beta\gamma\delta$, $\alpha'\beta'\gamma'\delta'$, and $\alpha''\beta''\gamma''\delta''$ are the unique decompositions of the hypercube labels, where $\alpha'=\alpha'_1\alpha'_2$ and $\alpha''=\alpha''_1\alpha''_2\alpha''_3$. Since v is a sprout in T, our construction of the embedding implies that the prefix $\alpha'_1=\alpha''_1$ of v is also a prefix of u, and, hence, u and v agree in segment A. By construction, we have $|\alpha'_2|=|\alpha''_2|=|\alpha''_3|=2$, $|\beta'|, |\beta''|\leq 2$, and $|\delta'|=|\delta''|=3$. Hence, if v is mapped to a child of the root of T (corresponding to row a)) the labels u and v obviously differ in at most $|\alpha'|+|\beta'|+2+|\delta'|\leq 9$ bit positions.

On the other hand, if v is mapped to a grandchild of the root of T (corresponding to row b)), u and v differ in at most $|\alpha''_1\alpha''_2|+|\beta''|+2+|\delta''|\leq 11$ bit positions. In the

latter case, the dilation can be reduced to at most 9 if we choose an appropriate $\beta'' \in \{1, 01, 11\}$ and γ'' such that u and v differ in at most 2 positions in segment C. This can be done as follows. If the single 1 in γ belongs to segment A or A', then choose $\beta'' \in \{1, 01\}$ and $\gamma \in 0^*10^*$ arbitrarily. If the single 1 in γ belongs to segment B, then choose $\gamma'' \in 0^*10^*$ arbitrarily and choose $\beta'' \in \{1, 01, 11\}$ such that the labels differ in at most one position in segment B. If the single 1 in γ belongs to segment C, choose $\beta'' \in \{1, 01, 11\}$ arbitrarily and choose γ'' such that it agrees with γ in segment C. Note that in every case there are at least two different possibilities to choose β'' from $\{1, 01, 11\}$.

5.4 Dilation of the Dynamic Embedding

Now we are ready to prove an upper bound on the dilation of the proposed embedding. Because of Lemma 7, it is sufficient to show that the load of each (h, o, τ)-tree node is bounded by its capacity. As we will see, this is true for each (h, o, τ)-tree node at level at most $h-6$. For nodes at a higher level, a sophisticated modification of our embedding will ensure that we obtain an embedding with dilation of at most 9.

In order to compute the number of tree vertices mapped to a single (h, o, τ)-tree node, let $n(\ell)$ be the maximum number of tree vertices mapped to a single node of the (h, o, τ)-tree at level ℓ. Because of our construction, we count in $n(\ell)$ three different type of tree vertices: marked vertices (i.e., neighbors of already mapped tree vertices), vertices incident to a edge removed by Lemma 8, and sprouts. Furthermore, let $f(\ell)$ be the size of the associated forest which is partitioned at a node of the (h, o, τ)-tree at level ℓ. An obvious upper bound for $f(\ell)$ is $2^{2h+3+\tau} - c(1)/4^{\ell-1}$. Recall that $c(\ell)$ denotes the capacity of a node in the (h, o, τ)-tree at level ℓ. The number of marked vertices in a forest corresponding to a node at level ℓ before the partitioning is at most $2c(\ell)$, since each mapped vertex has at most two unmapped neighbors. As described earlier, the sprouts in a subtree are distributed evenly among the four children of its root. Hence, by Lemma 8 we obtain for $\ell \geq 2$:

$$n(\ell) \;\leq\; 3\,\lfloor \log\left(f(\ell-1)\right)\rfloor + \left\lceil \frac{2c(\ell-1)}{4} + \frac{t(\ell-1)}{4} \right\rceil \qquad (1)$$

Note that in the special case of $\ell = 2$, there are no sprouts to distribute, because there are no sprouts in the whole (h, o, τ)-tree. On the other hand, because of the irregularity of the capacity of the root, we have also to distribute the neighbors of the tree vertices mapped to hypercube locations in R. Clearly, both terms can be bounded by $\frac{2^{o+1}}{4}(2(h-(\ell-1)+1)+1+\tau)$ in Equation 1, yielding the following bound on $n(\ell)$:

$$n(\ell) \;\leq\; 3\,\lfloor \log\left(f(\ell-1)\right)\rfloor + \left\lceil \frac{1}{4}\Big[2c(\ell-1) + t(\ell-1)\Big] \right\rceil$$

$h(\ell)$	$c(\ell)$	$n(\ell)$	$s(\ell)$	$\Delta(\ell)$	$\hat{t}(\ell)$	$\mu(\ell)$
7	296	296	0	0	68	226
6	248	256	0	8	60	196
5	200	218	2	20	52	166
4	152	180	5	23	44	136
3	104	142	6	44	36	106
2	56	(88)	11	(32)		
1	8	(8)	8	(0)		

a) $\tau = 0$

$h(\ell)$	$c(\ell)$	$n(\ell)$	$s(\ell)$	$\Delta(\ell)$	$\hat{t}(\ell)$	$\mu(\ell)$
7	320	313	0	0	72	241
6	272	275	0	3	64	211
5	224	237	1	14	56	181
4	176	199	4	27	48	151
3	128	161	7	38	40	121
2	80	123	10	53	32	91
1	32	(32)	11	(0)		

b) $\tau = 1$

$h(\ell)$=$(h-\ell+1)$ is the height of a node at level ℓ,
$c(\ell)$=$48h(\ell)-40+24\tau$ is the capacity of a node at level ℓ,
$n(\ell)$=$38h(\ell)+28+19\tau$ is the maximal number of vertices mapped to a node at level ℓ,
$s(\ell)$=$\lceil \frac{1}{4}\Delta(\ell-1) \rceil$ is the maximal number of vertices shifted downwards to a node at level ℓ,
$\Delta(\ell)$= $\max\{0, n(\ell)-c(\ell)+s(\ell)\}$ is the maximal excess at a node at level ℓ,
$\hat{t}(\ell)$=$8h(\ell)+12+4\tau$ is the number of sprouts counted in $n(\ell)$,
$\mu(\ell)$=$30h(\ell)+16+15\tau$ is the number of wooden tree vertices counted in $n(\ell)$.
(See also explanation in the text.)

Figure 6: $(h, 3, \tau)$-Tree Statistics

$$
\begin{aligned}
&\leq 3\left\lfloor \log\left(\frac{2^{2h+3+\tau}-1}{4^{\ell-2}}\right)\right\rfloor \\
&\quad + \frac{2^3}{2}\Big(6(h-(\ell-1)+1)-5+3\tau\Big) + \delta_{\ell,2}\frac{2\cdot|R|}{4} + (1-\delta_{\ell,2})\frac{t(\ell-1)}{4} \\
&\leq 3\,(2h-2\ell+6+\tau) \\
&\quad + 4\Big(6(h-\ell+1)+1+3\tau\Big) + \frac{2^{3+1}}{4}\Big(2(h-(\ell-1)+1)+1+\tau\Big) \\
&\leq 6(h-\ell+1)+12+3\tau \\
&\quad + 24(h-\ell+1)+4+12\tau+8(h-\ell+1)+12+4\tau \\
&= 38(h-\ell+1)+28+19\tau
\end{aligned}
$$

Since the capacity of a node at level ℓ of a $(h, 3, \tau)$-tree is $48(h-\ell+1)-40+24\tau$, $n(\ell)$ should be less than or equal this expression. This is the case for $\ell \leq h-6$.

Now, we take a closer look at the higher numbered levels, as given in Figure 6. The conflict that more vertices get mapped to a (h, o, τ)-tree node than its capacity allows will be resolved by shifting the excess downwards in the (h, o, τ)-tree. In what follows, we denote by $\Delta(\ell)$= $\max\{0, n(\ell)-c(\ell)+s(\ell)\}$ the excess at a (h, o, τ)-tree node at level ℓ and by $s(\ell)$=$\lceil \frac{1}{4}\Delta(\ell-1) \rceil$ the maximal number of vertices shifted to a (h, o, τ)-tree node at level ℓ. Furthermore, we denote by $\hat{t}(\ell)$ the maximal number of sprouts contained in a subtree rooted at a (h, o, τ)-tree node at level ℓ. The improved numbers in parentheses stem from the fact that because of our construction the number of tree

vertices mapped to a subtree of a (h, o, τ)-tree cannot exceed its overall capacity.

The excess $\Delta(\ell)$ at a (h, o, τ)-tree node will be distributed evenly among its four children. To maintain a small dilation, in a first step only sprouts will be shifted downwards. Since $\Delta(\ell) \leq \hat{t}(\ell)$ for $\ell \leq h-3+\tau$, this will resolve our conflict except for (h, o, τ)-tree nodes at level of at most $h-3+\tau$. This case will be considered later. Sprouts which are itself marked or incident to a removed edge will not be shifted. Note that in this case the sprout itself has been counted at least twice in $n(\ell)$ and we already have one fewer vertex to shift. Hence, a shifted sprout has exactly one already mapped neighbor, namely its parent. As we have seen in the previous subsection, we can choose hypercube locations for the shifted sprouts such that the dilation of the incident edge is at most 9. Since it might be possible that we have to choose for all shifted sprouts the same bit string for γ'', this strategy is successful only if there exists enough hypercube locations to which shifted sprouts can be mapped. As we mentioned earlier, there are at least two possibilities to choose β'' and since we made no restrictions on δ'', our strategy is successful if $s(\ell) \leq 2^{o+1}=16$. As can be seen from Figure 6, this is the case for all levels.

It remains to select vertices to shift downward from level $\ell \geq h-2+\tau$ to a node at level $\ell+1$. For this additional vertices we select tree vertices which have exactly one already mapped neighbor. More precisely, consider a forest associated with a (h, o, τ)-tree node at level ℓ. A tree vertex v of this forest is called a *single* tree vertex if one of the following conditions is fulfilled: v is a marked vertex which is not incident to a removed edge and has exactly one already mapped neighbor; or v is incident to exactly one removed edge and is not marked. Therefore, we choose only single tree vertices downwards. Note that a tree vertex which is not single but marked or incident to a removed tree edge is counted at least twice in $n(\ell)$ and again one fewer vertex have to be shifted downwards. We call such a tree vertex counted in $n(\ell)$ but not really existent a *phantom* vertex.

Let $\mu(\ell)=30(h-\ell+1)+16+15\tau$ denote the number of tree vertices counted in $n(\ell)$ which are marked or incident to a removed edge. First, we observe that at least $\frac{1}{2}\mu(\ell)$ of these tree vertices in $n(\ell)$ are either single or phantom tree vertices. Since $\Delta(\ell) \leq \hat{t}(\ell)+\frac{1}{2}\mu(\ell)$, there are sufficiently many single tree vertices for shifting downwards except in the case $(\ell, \tau)=(h-1, 0)$ whose solution is postponed at the end of this subsection.

To estimate the dilation of the tree edges incident to shifted marked tree vertices, see Figure 7. Here, v denotes the shifted marked vertex and u its already mapped neighbor. Row a) corresponds to the situation in which u is not shifted. Row b) corresponds to the case in which u is also shifted downwards. Finally, row c) reflects the case when u and v are incident to a removed edge and both vertices are shifted. Note that $\alpha\beta\gamma\delta$, $\alpha'\beta'\gamma'\delta'$, $\alpha''\beta''\gamma''\delta''$, and $\alpha'''\beta'''\gamma'''\delta'''$ are the unique decompositions of the labels where $\alpha=\alpha_1\alpha_2\alpha_3$, $\alpha''=\alpha_1''\alpha_2''$, and $\alpha'''=\alpha_1'''\alpha_2'''\alpha_3'''$.

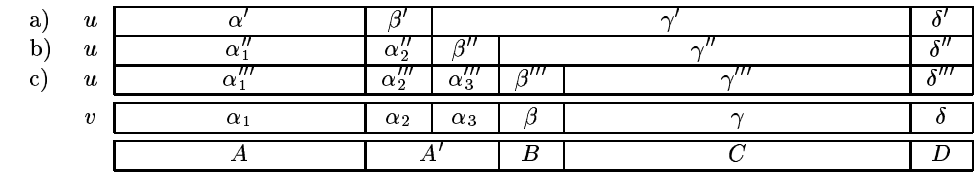

Figure 7: Possible Hypercube Locations of a Shifted Vertex v and Its Neighbor u

Again, we make a case analysis depending on the position of the single 1 in γ', γ'', or γ''', respectively. If the the single 1 belongs to segment A', we chose $\beta \in \{1, 01\}$ and $\gamma \in 0^*10^*$ arbitrarily. Otherwise, if the single one corresponds to segment B, $\gamma \in 0^*10^*$ is chosen arbitrarily and $\beta \in \{1, 01, 11\}$ is chosen such that u and v differ in segment B in at most one bit position. Finally, if the single 1 belongs to segment C, $\beta \in \{1, 01, 11\}$ is chosen arbitrarily and γ is chosen such that u and v agree in segment C. Note again that there are at least $2 \cdot 2^o = 16$ different choices for each vertex v. Since the number of shifted tree vertices to a node at level ℓ is at most $s(\ell) \leq 16$, the proposed strategy will resolve the problem that more tree vertices are mapped to a (h, o, τ)-tree nodes as their capacities allow except for the case $(\ell, \tau) = (h-1, 0)$.

Fortunately, in the case $(\ell, \tau) = (h-1, 0)$ the problem will disappear which can be seen as follows. Consider a subtree T of the (h, o, τ)-tree rooted at a node at level $h-2$. Because of our construction, all neighbors of vertices initially mapped to a node at level $h-1$ or h are mapped to T except the sprouts in T. By Lemma 3, T is itself a $(3, o, 0)$-tree and, therefore by Lemma 6, it is contained in an 9-dimensional subcube. This implies that if we shift only wooden vertices from level $h-1$ to level h, all edges incident to vertices mapped to T have dilation at most 9.

In this subsection, we have shown that it is possible to construct a dynamic embedding of binary trees into hypercubes with small dilation. Until now, we have specified the mapping of the vertices only. As mentioned earlier, it is necessary to specify also the paths connecting the images of endpoints of all edges to obtain a small node-congestion. For the details, we refer the reader to [17], because the construction is the same as in the case of static embeddings. The choice of the paths is quite simple although its analysis is lengthy and tedious. Note that the expansion of this embedding is less than 2 and, therefore, optimal. In the following subsections, we will allow a slightly larger expansion of $2+\varepsilon$ for any $\varepsilon > 0$ to obtain an efficient construction of this embedding.

5.5 Amortized Time Complexity for a Single New Leaf

In this subsection, we will bound the amortized cost for each newly spawned leaf. For the analysis of the amortized time complexity of our algorithm, we first assume

that in each spawning step exactly one new leaf is grown. We will later generalize our results assuming that in each step at most M new children get spawned.

In what follows, we describe a dynamic embedding with expansion $2+\varepsilon$, for any $\varepsilon \geq 0$. Recall that any dynamic embedding with unit load must have expansion of nearly 2 in the worst case, because hypercubes are only available for powers of two. Therefore, the presented dynamic embedding is nearly optimal with respect to the expansion. Thus, we consider the growing period of the binary tree in which its size increases from $2^{2h+3+\tau}/(2+\varepsilon)$ to $2^{2h+3+\tau}/(1+\varepsilon/2)$. The tree grows therefore in a $2h+3+\tau$-dimensional hypercube or, equivalently, in an $(h, 3, \tau)$-tree.

The *load of a subtree* T of a (h, o, τ)-tree, denoted by $\mathrm{load}(T)$, is the sum of the loads over all vertices in T. Remember that a subtree of a (h, o, τ)-tree is itself a (h, o, τ)-tree. The *slack of a subtree* T of a (h, o, τ)-tree is defined as the overall capacity of T minus the load of T. Consider a subtree T of the (h, o, τ)-tree rooted at node v and let T_j, for $j \in [1{:}4]$, be the four subtrees rooted at the children of v. We denote by

$$\mathrm{imb}(T) = \sum_{i=1}^{4} \left[\mathrm{load}(T_i) - \min_{j \in [1:4]} (\mathrm{load}(T_j)) \right]$$

the *imbalance* of the subtree T.

A first important observation is that a recomputation of an embedding in a subtree T of a (h, o, τ)-tree reduces its imbalance to at most 3. Hence, the imbalance of a subtree T implies that after the last rebalancing in which the whole tree was involved at least $\mathrm{imb}(T)-3$ tree vertices have been mapped to T. Furthermore, if the load of a subtree T is at most its overall capacity then the imbalance of T is bounded by the number of new leaves spawned in a single spawning step.

We now give a more detailed description of the migration step. After the spawning step, we first remap all new leaves to leaves in the (h, o, τ)-tree as described earlier. Then, we recompute the embedding in all subtrees of a small constant height, say three. For each subtree of height 3 whose load exceeds its overall capacity, we call the procedure rebalance from its root. Invoked at a node v, the procedure rebalance determines the subtree of the (h, o, τ)-tree containing the node v, in which the embedding will be recomputed, cf. Figure 8. More precisely, consider a (h, o, τ)-tree node v at which the procedure rebalance is invoked. If the load of the subtree rooted at node v is greater than its capacity or the imbalance of this subtree is less than $\kappa(h')^5$ then we call the procedure rebalance recursively with the parent of v as its argument. Here, κ is a constant depending on ε to be defined later. Otherwise we compute the embedding for the subtree as mentioned in the previous section. If the procedure rebalance is invoked from the parent of the root of the whole (h, o, τ)-tree, the hypercube grows by one dimension and the the whole embedding will be recomputed using the next larger (h, o, τ)-tree.

```
procedure rebalance(v);
    if (load(T(v))>capacity(T(v))) or (imbalance(T(v))< κ·height(T(v))^5)
        then
            rebalance(parent(v))
        else
            embed_statically(T(v))
    endif;
end rebalance;
```

Figure 8: The Procedure Rebalance

After the migration step, we have to ensure that the load of each subtree is at most its overall capacity. Since we only use the embedding algorithm for a subtree of the (h, o, τ)-tree whose load is at most its overall capacity, we achieve an embedding of the binary tree into the hypercube with unit load. In what follows, we show that the load of the (h, o, τ)-tree is at least a fraction of $1/(1+\varepsilon/2)$ of its overall capacity if the procedure rebalance is invoked at the parent of the root of the (h, o, τ)-tree.

Suppose that the procedure rebalance is invoked at the parent of the root. This implies that either the load of the (h, o, τ)-tree is greater than its overall capacity or the imbalance at the root is less than $\kappa \cdot h^5$. In the first case we are done. Now assume that the imbalance of the (h, o, τ)-tree is less than $\kappa \cdot h^5$. We will count the slack of the (h, o, τ)-tree. The first call to the procedure rebalance during this rebalancing was made by a leaf of the (h, o, τ)-tree with zero slack. At each node v on the path from the leaf to the root, the imbalance of the subtree rooted at v is less than $\kappa(h-\ell+1)^5$, where ℓ is the level of v in the (h, o, τ)-tree. Thus, we get the following recurrence for the slack $S(\ell)$ of the subtree rooted at a vertex at level ℓ on that path:

$$S(h) = 0; \qquad S(\ell) \leq 4S(\ell+1) + 3\kappa(h-\ell+1)^5.$$

It can easily be verified that

$$S(\ell) \leq 3 \sum_{i=\ell}^{h-1} 4^{i-\ell} \kappa (h-i+1)^5$$

is a solution of this recurrence. Thus, we obtain

$$\begin{aligned} S(1) &\leq 3 \sum_{i=0}^{h-2} 4^i \kappa (h-i)^5 \\ &= 3\kappa \sum_{i=2}^{h} 4^{h-i} i^5 \\ &= 3\kappa 4^h \sum_{i=2}^{h} \frac{i^5}{4^i} \\ &\quad \text{since } \textstyle\sum_{i=2}^{\infty} \frac{i^5}{4^i} \leq 17 \end{aligned}$$

$$\begin{aligned} &\leq 51\kappa 2^{2h} \\ &\quad \text{choosing } \kappa = \tfrac{8\varepsilon}{51(2+\varepsilon)} \leq \tfrac{2^{3+\tau}\varepsilon}{51(2+\varepsilon)} \\ &\leq \frac{\varepsilon}{2+\varepsilon} 2^{2h+3+\tau} \end{aligned}$$

Hence, the slack of the (h,o,τ)-tree is at most $\frac{\varepsilon}{2+\varepsilon}$ of its overall capacity, implying that the load of the (h,o,τ)-tree is at least $1-\frac{\varepsilon}{2+\varepsilon}=\frac{2}{2+\varepsilon}$ of its overall capacity. Whenever the procedure rebalance is invoked at the parent of the root, we increase the dimension of the hypercube by one. Thus, the load of the larger (h,o,τ)-tree is still at least $\frac{1}{2}\cdot\frac{2}{2+\varepsilon}=\frac{1}{2+\varepsilon}$ of its overall capacity, implying that the expansion of the embedding is at most $2+\varepsilon$.

We will now compute the amortized cost for embedding a new leaf. As mentioned earlier, the imbalance of a subtree T implies that at least $\max\{\mathrm{imb}(T)-3,0\}$ tree vertices have been mapped to T since the last rebalancing in which the whole tree was involved. Suppose that the procedure rebalance is invoked at node v and that the level of v in the (h,o,τ)-tree is ℓ. As stated in the previous section in Lemma 9, the recomputation of the embedding can be done in time $O((h-\ell+1)^3)$. Since the imbalance of the subtree rooted at v is at least $\kappa\cdot(h-\ell+1)^5$, we charge to each of these tree vertices an amount of $O(1/\kappa(h-\ell+1)^2)$. Note further that each tree vertex can contribute at most once to the imbalance of a subtree rooted at a node v at some fixed level, since after the rebalancing of the subtree rooted at v the imbalance of that tree is at most 3. Hence, we charge to each vertex an amount of at most

$$O\left(\frac{1}{\kappa}\sum_{\ell=1}^{h}\frac{1}{(h-\ell+1)^2}\right) \leq O\left(\frac{\pi^2}{6\kappa}\right) = O(\varepsilon^{-1}).$$

Thus, the amortized cost for embedding a newly spawned tree vertex is at most $O(\varepsilon^{-1})$.

Theorem 12 *For any $\varepsilon>0$, an arbitrary binary tree can be dynamically embedded into a hypercube with unit load, dilation of at most 9, constant node-congestion, and expansion of at most $2+\varepsilon$. The amortized time complexity for each spawning step of the embedding is $O(\varepsilon^{-1})$ provided that in each spawning step at most one new leaf is spawned.*

Choosing $\varepsilon=2$ we obtain the following corollary.

Corollary 13 *An arbitrary binary tree can be dynamically embedded into its next to optimal hypercube with unit load, dilation of at most 9, and constant node-congestion. The embedding can be computed on the hypercube in constant amortized time provided that in each spawning step at most one new leaf is spawned.*

5.6 Amortized Time Complexity for a Group of New Leaves

We now consider the case that in each spawning step at most L new leaves are grown. It is possible that all these leaves are spawned from tree vertices which are mapped to a subtree of the (h, o, τ)-tree of height $\Theta(\log_4(L))$. Because of our construction they cannot be mapped into a subtree of height $o(\log_4(L))$. After each spawning step, we recompute the embedding in each subtree of the (h, o, τ)-tree of height $2\alpha \log_4(L) = \alpha \log(L)$ and invoke the procedure rebalance at its root if the load of the subtree exceeds its overall capacity. Here, α is some constant to be chosen later. We also modify our rebalancing procedure as follows. The procedure rebalance will now be invoked from a node v at level ℓ if the imbalance of the subtree rooted at v is at least $\kappa \cdot L(h-\ell+1)^5$.

Since we recompute the embedding in a subtree after each spawning step, we have to ensure that it can be computed quickly. In order to obtain a more efficient algorithm, we store for each subtree of the (h, o, τ)-tree of height $\alpha \log(L)$ also the Euler contour path linearly in the corresponding subcube. Because of Theorem 10, it is then possible to compute the reembedding in time $O(\alpha^2 \log^2(L))$. Note that we simply recompute the Euler contour path if a reembedding in a subtree of height $h > \alpha \log(L)$ is necessary. Clearly, this can be done within the reembedding in time $O(h^3)$. If we recompute the a subtree of height $\alpha \log_4(L)$, it is necessary to update the linearly stored Euler contour path. Note that spawning a new leaf corresponds to a insertion of three consecutive list vertices in the Euler contour path. Using Batcher's sort [1, 24], the information about the newly spawned vertices can be easily routed within the subcube from the spawning parents to its corresponding list vertices in the Euler contour path in time $O(\log^2(L))$. Using parallel prefix operations [26], the modified Euler contour path can be restored linearly in time $O(\log(L))$. Thus, the remapping after each spawning step can be done in time $O(\log^2(L))$.

To determine the amortized running time, we distribute again the cost for computing the embedding to the tree vertices which contributes to the imbalance. Assume the procedure rebalance constructs a reembedding into a subtree rooted at a node at level ℓ where $(h-\ell+1) > \alpha \log(L)$. By Theorem 9, the computation time is bounded by $O((h-\ell+1)^3)$. Now, we charge the amount of $O(1/(\kappa \cdot L(h-\ell+1)^2))$ to each tree vertex which contributes to the imbalance. Thus, we charge to each spawning step the amount of at most

$$O\left(L \sum_{\ell=1}^{h-\alpha\log(L)} \frac{\kappa^{-1} \cdot L^{-1}}{(h-\ell+1)^2}\right) \leq O\left(\kappa^{-1} \int_{\alpha\log(L)}^{\infty} \frac{dx}{x^2}\right) \leq O\left(\frac{\kappa^{-1}}{\alpha \log(L)}\right)$$

Thus, the total amortized computation time for each group of at most L newly spawned leaves is $O(\alpha^2 \log^2(L) + \kappa^{-1}/(\alpha \log(L)))$. For the following analysis, we need a technical lemma.

Lemma 14 *For $k \leq a \in \mathbb{N}$, we have* $\displaystyle\sum_{i=a}^{\infty} \frac{i^k}{4^i} \leq 4 \cdot \frac{a^k}{4^a}$.

Proof: Obviously, for $k \leq a$, we have

$$k \ln\left(\frac{i+a}{a}\right) \leq k \ln\left(1+\frac{i}{a}\right) \leq k\frac{i}{a} \leq i,$$

since $\ln(1+x) \leq x$ for $x \geq 0$. Therefore, we get $((i+a)/a)^k \leq e^i$. Thus,

$$\sum_{i=a}^{\infty} \frac{i^k}{4^i} \cdot \frac{4^a}{a^k} = \sum_{i=0}^{\infty} \frac{(\frac{i+a}{a})^k}{4^i} \leq \sum_{i=0}^{\infty} \left(\frac{e}{4}\right)^i = \frac{1}{1-e/4} < 4,$$

finishing the proof. ∎

In what follows, we show that the load of the (h,o,τ)-tree is at least a fraction of $1/(1+\varepsilon/2)$ of its overall capacity if the procedure rebalance is called from the parent of the root. Again, we will count the slack of the (h,o,τ)-tree. Since we recompute the embedding of each subtree of height $\alpha \log(L)$ after each spawning step, we now obtain the following recurrence:

$$S(h - \alpha\log(L) + 1) = 0; \qquad S(\ell) \leq 4S(\ell+1) + 3\kappa L(h-\ell+1)^5.$$

Thus, we get the following solution of this recurrence:

$$S(\ell) \leq 3 \sum_{i=\ell}^{h-\alpha\log(L)} 4^{i-\ell}\kappa L(h-i+1)^5.$$

Hence, the slack of the (h,o,τ)-tree is bounded by:

$$\begin{aligned}
S(1) &\leq 3 \sum_{i=1}^{h-\alpha\log(L)} 4^{i-1}\kappa L(h-i+1)^5 \\
&= 3\kappa L \sum_{i=\alpha\log(L)+1}^{h} i^5 4^{h-i} \\
&\leq 3\kappa L 4^h \sum_{i=\alpha\log(L)}^{\infty} \frac{i^5}{4^i} \\
&\quad \text{using Lemma 14 under the assumption that } 5 \leq \alpha\log(L) \\
&\leq 12\kappa L 4^h \frac{(\alpha\log(L))^5}{4^{\alpha\log(L)}} \\
&\leq 12\kappa L^{1-2\alpha}(\alpha\log(L))^5 4^h \\
&\quad \text{choosing } \kappa = \frac{4L^{2\alpha-1}\varepsilon}{6(\alpha\log(L))^5(2+\varepsilon)} \leq \frac{2^{3+\tau}\varepsilon L^{2\alpha-1}}{12(\alpha\log(L))^5(2+\varepsilon)} \\
&\leq \frac{\varepsilon}{2+\varepsilon} 2^{2h+3+\tau}
\end{aligned}$$

Again, the load of the (h, o, τ)-tree is at least $\frac{\varepsilon}{2+\varepsilon}$ of its overall capacity. The same computation as before shows that after increasing the hypercube by one dimension, the overall load of the (h, o, τ)-tree is at least $\frac{1}{2+\varepsilon}$ of its overall capacity implying that the expansion of the constructed embedding is at most $2+\varepsilon$.

Theorem 15 *For any $\varepsilon>0$ and $\alpha>0$, an arbitrary binary tree can be dynamically embedded into a hypercube with unit load, dilation of at most 9, constant node-congestion, and expansion of at most $2+\varepsilon$. If in each step at most $L \geq 2^{5/\alpha}$ leaves are spawned, the embedding can be computed on the hypercube in amortized time $O(\alpha^2 \log^2(L)+\varepsilon^{-1}L^{1-2\alpha}\alpha^4 \log^4(L))$ per spawning step.*

In particular, we obtain the following corollary for $\alpha=3$:

Corollary 16 *For any $\varepsilon>0$, an arbitrary binary tree can be dynamically embedded into a hypercube with unit load, dilation of at most 9, constant node-congestion, and expansion of at most $2+\varepsilon$. The embedding can be computed on the hypercube in amortized time $O(\log^2(L)+\varepsilon^{-1}\log^4(L)/L^5)=O(\log^2(L)+\varepsilon^{-1})$ per spawning step, if in each step at most L new leaves are spawned.*

Note that the above corollary is true for all L. For small L a more careful analysis shows that a slightly smaller κ removes the bound on L. Choosing $\varepsilon=\log^2(L)/L^5$, we obtain the following corollary.

Corollary 17 *An arbitrary binary tree can be dynamically embedded into a hypercube with unit load, dilation of at most 9, constant node-congestion, and expansion of at most $2+\log^2(L)/L^5$. Furthermore, this embedding can be computed on the hypercube in amortized time $O(\log^2(L))$ per spawning step, if in each step at most L leaves are spawned.*

5.7 Remarks on the Implementation

Recall that we have assumed that there exist spawning steps and migration steps which alternate. This assumption is not correct. Consider two small subtrees of the (h, o, τ)-tree which decided to reembed their embeddings. The distance of these two subtrees in the hypercube might be larger than the computation needed for the reembedding of both subtrees. Thus, it is in general not possible to broadcast to the whole (h, o, τ)-tree that there is a migration step. So in general, some parts of the binary tree will spawn new leaves while some other parts of the binary tree are involved in a reembedding phase. Fortunately, this will not effect our algorithm. Since disjoint subtrees of a (h, o, τ)-tree are stored in disjoint subcubes of the hypercube,

a reembedding in one subcube cannot interact with a spawning or reembedding step in another subcube. Moreover, the algorithm for dynamic embedding will even run faster than our complexity analysis shows.

Another problem which may arise is that in one subtree of a (h, o, τ)-tree the rebalancing procedure will climb the (h, o, τ)-tree upwards while another subtree whose root is successor of a vertex in the rebalancing path is already involved in a reembedding phase. Thus, the reembedding phase of two subtrees might overlap. But, of course, this problem can be easily solved. First a message is broadcasted to the whole subtree of the (h, o, τ)-tree when a reembedding is necessary. A root of a subtree which receives this message and which had itself initiated a reembedding of its subtree broadcasts to their successors that the reembedding is interrupted. Clearly, the time for broadcasting is negligible against the time for the reembedding.

6 Dynamic Embedding of Large Binary Trees

In this section, we extend our algorithm to obtain a dynamic embedding for large binary trees with nearly optimal load. Although its possible to embed a binary tree with optimal load, the slack in the load allows us once more to compute the embedding efficiently.

6.1 Modifications for Embeddings with High Load

In the following, we construct an embedding of a large binary tree of size M into a smaller hypercube of size N. Again the embedding is based on our main tool, the (h, o, τ)-tree. Here, we will use a $(h, 0, \tau)$-tree to obtain an embedding with nearly optimal load. The achieved load of our embedding will be denoted by ρ. We choose ρ slightly larger than the optimal load $\lceil M/N \rceil$ to obtain an efficient construction of the dynamic embedding.

To construct an embedding with load ρ, we allow in the first embedding step that a $(h, 0, \tau)$-tree node at level ℓ gets at most $\rho \cdot c(\ell)$ tree vertices. We call $\rho \cdot c(\ell)$ the *extended capacity* of a node at level ℓ. It is easy to see how the second step of our embedding has to be modified to obtain an embedding with load ρ. In the sequel, we will prove that, for any $\varepsilon > 0$, it is possible to dynamically embed a binary tree of size M into a hypercube of size N with load $\lfloor (1+\varepsilon)\lceil M/N \rceil \rfloor$ and dilation of at most 6. Note that the load is a factor of $(1+\varepsilon)$ within the optimal load. This implies that we consider a $(h, 0, \tau)$-tree with $\tau = \lceil \log(M/\rho) \rceil \bmod 2$ and $h = \lceil \frac{1}{2}(\log(M/\rho) - \tau) \rceil$.

In what follows, we consider the size N of the hypercube as fixed and we adjust the load ρ of the embedding only if necessary. It is also possible to consider the load ρ

as fixed while varying the size N of the used hypercube. In the latter case, we get analogous results and the analysis is similar to the analysis given in the previous section.

6.2 Dilation of the Embedding

Now we are ready to prove the bound on the dilation of the proposed embedding with load of at most ρ. Again, it is sufficient to show that the load of each $(h,0,\tau)$-tree node is bounded by its extended capacity.

Again, let $n(\ell)$ be the maximum number of tree vertices mapped to a single node of the $(h,0,\tau)$-tree at level ℓ. Furthermore, let $f(\ell)$ be the size of the associated forest which is partitioned at a node of the $(h,0,\tau)$-tree at level ℓ. An obvious upper bound for $f(\ell)$ is now $\rho(2^{2h+\tau}-c(1))/4^{\ell-1}$. The number of marked vertices in a forest corresponding to a node at level ℓ before the partitioning is at most $2\rho\cdot c(\ell)$. As described earlier, the sprouts in a subtree are distributed evenly among the four children of its root. Therefore, by Lemma 8 we obtain for $\ell\geq 2$:

$$\begin{aligned}
n(\ell) &\leq 3\lfloor\log(f(\ell-1))\rfloor + \left\lceil\frac{1}{4}\Big[2c(\ell-1)+t(\ell-1)\Big]\right\rceil \\
&\leq 3\left\lfloor\log\left(\frac{\rho(2^{2h+\tau}-1)}{4^{\ell-2}}\right)\right\rfloor \\
&\quad + \frac{\rho}{2}\Big(6(h-(\ell-1)+1)-5+3\tau\Big) + \delta_{\ell,2}\frac{2\rho\cdot|R|}{4} + (1-\delta_{\ell,2})\frac{\rho\cdot t(\ell-1)}{4} \\
&\leq 3\,(2h-2\ell+3+\tau+\lfloor\log(\rho)\rfloor) \\
&\quad + \frac{\rho}{2}\Big(6(h-\ell+1)+1+3\tau\Big) + \frac{\rho}{2}\Big(2(h-(\ell-1)+1)+1+\tau\Big) \\
&\leq 6(h-\ell+1)+3+3\tau+\rho \\
&\quad + 3\rho(h-\ell+1) + \rho\frac{1+3\tau}{2} + \rho(h-\ell+1) + \rho\frac{3+\tau}{2} \\
&= (4\rho+6)(h-\ell+1)+3\rho+3+\tau(2\rho+3)
\end{aligned}$$

Since the extended capacity of a $(h,0,\tau)$-tree node at level ℓ is $\rho(6(h-\ell+1)-5+3\tau)$, $n(\ell)$ should be less than or equal this expression. For $\rho\geq 16$, this is the case for $\ell\leq h-5$.

Now, we take a closer look at the higher numbered levels, as given in Figure 6. Using the same modifications of the embedding as in the previous section, it is possible to resolve the conflict that more tree vertices are mapped to a $(h,0,\tau)$-tree node than allowed by its extended capacity. Again some appropriate chosen tree vertices are shifted downwards in the $(h,0,\tau)$-tree. Since $\Delta(\ell)\leq\hat{t}(\ell)+\frac{1}{2}\mu(\ell)$, this modification is successful. From the bound on the dilation of the one-to-one embedding, it follows that the dilation of this embedding is at most 6. Note that no assumptions were

$h(\ell)$	$\bar{c}(\ell)$	$n(\ell)$	$s(\ell)$	$\Delta(\ell)$	$\hat{t}(\ell)$
6	31ρ	$27\rho{+}39$	0	0	$15/2\rho$
5	25ρ	$23\rho{+}33$	0	≤ 1	$13/2\rho$
4	19ρ	$19\rho{+}27$	≤ 1	≤ 28	$11/2\rho$
3	13ρ	$15\rho{+}21$	≤ 7	$\leq 2\rho{+}28$	$9/2\rho$
2	7ρ	(11ρ)	$<\rho$	$\leq(4\rho)$	
1	ρ	(ρ)	$\leq(\rho)$	(0)	

a) $\tau = 0$

$h(\ell)$	$\bar{c}(\ell)$	$n(\ell)$	$s(\ell)$	$\Delta(\ell)$	$\hat{t}(\ell)$	$\mu(\ell)$
6	34ρ	$29\rho{+}42$	0	0	8ρ	21ρ
5	28ρ	$25\rho{+}36$	0	0	7ρ	18ρ
4	22ρ	$21\rho{+}30$	0	≤ 14	6ρ	15ρ
3	16ρ	$17\rho{+}24$	≤ 4	$\leq\rho{+}28$	5ρ	12ρ
2	10ρ	$13\rho{+}18$	$<\rho{-}2$	$<5\rho$	$<4\rho$	9ρ
1	4ρ	(4ρ)	$<5/4\rho$			

b) $\tau = 1$

$h(\ell)=(h-\ell+1)$ is the height of a node at level ℓ,
$\bar{c}(\ell)=\rho\cdot c(\ell)=\rho(6h(\ell)-5+3\tau)$ is the extended capacity of a node at level ℓ,
$n(\ell)=(4\rho+6)h(\ell)+3\rho+3+\tau(2\rho+3)$ is the maximal number of vertices mapped to a node at level ℓ,
$s(\ell)=\lceil\frac{1}{4}\Delta(\ell-1)\rceil$ is the maximal number of vertices shifted downwards to a node at level ℓ,
$\Delta(\ell)=\max\{0, n(\ell)-\bar{c}(\ell)+s(\ell)\}$ is the maximal excess at a node at level ℓ,
$\hat{t}(\ell)=\rho\cdot h(\ell)+\rho\frac{3+\tau}{2}$ is the number of sprouts counted in $n(\ell)$,
$\mu(\ell)=(3\rho+6)h(\ell)+\rho\frac{3(1+\tau)}{2}+3(1+\tau)$ is the number of the wooden tree vertices counted in $n(\ell)$,
and $\rho\geq 16$ is the load of the embedding.
(See also explanation in the text.)

Figure 9: $(h, 0, \tau)$-Tree Statistics for Embeddings with Load ρ

made about the last o bit positions in the analysis of the dilation of the one-to-one embedding in the previous section.

Theorem 18 *For any $\rho\geq\max\{16, \lceil M/N\rceil\}$, an arbitrary binary tree of size M can be embedded into a hypercube of size N with load ρ and dilation of at most* 6.

6.3 Amortized Time Complexity

In this subsection, we analyze the amortized time required for our dynamic embedding. First, we determine the running time for our static embedding algorithm under the assumption that the considered tree is larger than the used hypercube. A closer inspection of the construction in [17] shows that the static embedding of a binary tree of size M on a hypercube of size N can be computed in time $O(M/N\cdot\log^3(N))$. If the Euler contour path of the binary tree is already stored linearly, the running time can be improved to $O(M/N\cdot\log^2(N))$. Now consider a reembedding with load ρ in a subtree rooted at a node at level ℓ. Hence, this recomputation can be performed in time $O(\rho(h-\ell+1)^3)$. Provided that the Euler contour path is stored linearly, the recomputation can be done in time $O(\rho(h-\ell+1)^2)$.

In the following, let L be the number of new leaves in each spawning step and ρ the actual load of the embedding. We will recompute after each spawning step the

embedding in subtrees of the $(h,0,\tau)$-tree of height $\alpha\log(L/\rho)$ for some constant α. If L is a small with respect to ρ, the embedding is computed for subtrees of small constant height only. For recomputations of subtrees rooted at a node at level ℓ where $(h-\ell+1)>\alpha\log(L/\rho)$, we recurse using the procedure rebalance as explained in the previous section. In the procedure rebalance, the bound on the imbalance is again chosen as $\kappa\cdot L(h-\ell+1)^5$. As mentioned above, the computation time for a reembedding into a subtree of height $h-\ell+1>\alpha\log(L)$ is bounded by $O(\rho(h-\ell+1))^3)$. We charge the amount of $O(\rho/(\kappa\cdot L(h-\ell+1)^2))$ to each tree vertex which contributes to the imbalance. Thus, we charge to each spawning step the amount of at most

$$O\left(L\cdot\frac{\rho}{\kappa\cdot L}\sum_{\ell=1}^{h-\alpha\log(L/\rho)}\frac{1}{(h-\ell+1)^2}\right)=O\left(\frac{\rho}{\kappa}\int_{\alpha\log(L/\rho)}^{\infty}\frac{dx}{x^2}\right)=O\left(\frac{\rho}{\kappa\cdot\alpha\log(L/\rho)}\right).$$

Hence, the total amortized computation time for each group of at most L newly spawned leaves is

$$O\left(\rho\cdot\alpha^2\log^2(L/\rho)+\frac{\rho}{\kappa\cdot\alpha\log(L/\rho)}\right)$$

Note that, for convenience, we denote by $\log(x)$ the function $\max\{1,\log_2(x)\}$.

In what follows, we show that the load of the $(h,0,\tau)$-tree is at least a fraction of $\varepsilon/(2+2\varepsilon)$ of its overall extended capacity if the procedure rebalance is called from the parent of the root. Again, we will count the slack of the $(h,0,\tau)$-tree. Since we recompute the embedding of each subtree of height $\alpha\log(L/\rho)$ after a spawning step, we obtain once more the following recurrence:

$$S(h-\alpha\log(L/\rho)+1)=0;\qquad S(\ell)\le 4S(\ell+1)+3\kappa L(h-\ell+1)^5.$$

Thus, we get the following solution of this recurrence:

$$S(\ell)\le 3\sum_{i=\ell}^{h-\alpha\log(L/\rho)}4^{i-\ell}\kappa L(h-i+1)^5.$$

Hence, the slack of the $(h,0,\tau)$-tree is bounded by:

$$\begin{aligned}
S(1) &\le 3\sum_{i=1}^{h-\alpha\log(L/\rho)}4^{i-1}\kappa L(h-i+1)^5\\
&= 3\kappa L\sum_{i=\alpha\log(L/\rho)+1}^{h}4^{h-i}\cdot i^5\\
&\le 3\kappa 4^h L\sum_{i=\alpha\log(L/\rho)}^{\infty}\frac{i^5}{4^i}\\
&\quad \text{Using Lemma 14 assuming } \alpha\log(L/\rho)\ge 5\\
&\le 12\kappa 4^h L\frac{\alpha^5\log^5(L/\rho)}{(L/\rho)^{2\alpha}}
\end{aligned}$$

$$\begin{aligned} &\leq 12\rho \cdot 4^h \kappa \frac{\alpha^5 \log^5(L/\rho)}{(L/\rho)^{2\alpha-1}} \\ &\quad \text{choosing } \kappa = \frac{2^\tau (L/\rho)^{2\alpha-1} \varepsilon}{12\alpha^5 \log^5(L/\rho)(1+\varepsilon)} \\ &\leq \frac{\varepsilon}{2(1+\varepsilon)} \rho 2^{2h+\tau} \end{aligned}$$

Note that a similar computation holds if $\alpha \log(L/\rho) < 5$. This stems from the fact that $\sum_i i^5/4^i$ converges. Of course, in this case the value of κ will slightly differ from the value chosen above.

Hence, at least a fraction of $1 - \frac{\varepsilon}{2(1+\varepsilon)} = \frac{2+\varepsilon}{2(1+\varepsilon)}$ of the extended overall capacity of the $(h, 0, \tau)$-tree is used. If we now increase the load of the embedding from ρ to $\rho' = \lfloor (1+\varepsilon/2)\rho \rfloor$, we obtain:

$$\frac{M'}{N} \geq \frac{2+\varepsilon}{2(1+\varepsilon)} \cdot \rho \geq \frac{2+\varepsilon}{2(1+\varepsilon)} \cdot \frac{\rho'}{1+\frac{\varepsilon}{2}} = \frac{1}{1+\varepsilon} \rho'.$$

Here, M' denotes the size of the grown binary tree and $N = 2^{2h+\tau}$ denotes the size of the used hypercube. This implies that the load of the new embedding is within of $1+\varepsilon$ of the optimal load, i.e., $\rho' \leq \lfloor (1+\varepsilon) \lceil M/N \rceil \rfloor$. Altogether, we have proved the following theorem.

Theorem 19 *For any $\varepsilon > 0$ and $\alpha > 0$, an arbitrary binary tree of size M can be dynamically embedded into a hypercube of size N with load $\rho = \lfloor (1+\varepsilon) \lceil M/N \rceil \rfloor$ and dilation of at most 6, if $\rho \geq \max\{16, 2 \cdot \varepsilon^{-1}\}$. If in each spawning step at most L news are spawned then the amortized time complexity*

$$O\left(\rho\left(\alpha^2 \log^2(L/\rho) + \frac{\alpha^4 \log^4(L/\rho)}{\varepsilon (L/\rho)^{2\alpha-1}}\right)\right)$$

for each spawning step.

For instance, choosing $\alpha = 3$, we obtain the following corollary

Corollary 20 *For any $\varepsilon > 0$, an arbitrary binary tree of size M can be dynamically embedded into a hypercube of size N with load $\rho = \lfloor (1+\varepsilon) \lceil M/N \rceil \rfloor$ and dilation of at most 6, if $\rho \geq \max\{16, 2 \cdot \varepsilon^{-1}\}$. If in each spawning step at most L news are spawned then the amortized time complexity*

$$O\left(\rho\left(\log^2(L/\rho) + \frac{\log^4(L/\rho)}{\varepsilon (L/\rho)^5}\right)\right) = O(\rho(\log^2(L/\rho) + \varepsilon^{-1}))$$

for each spawning step.

Note that these theorems also hold in the case $L \leq \rho$, in which the amortized running time simplifies to $O(\rho/\varepsilon)$.

7 Conclusion

We have presented a deterministic algorithm for dynamically embedding arbitrary binary trees into hypercubes with unit load, dilation at most 9, constant edge- and node-congestion, and nearly optimal expansion. The algorithm can be implemented on the hypercube itself spending amortized time $O(\log^2(L))$ if in each step at most L new children are inserted. The necessity of the migration of tree vertices follows from a simple adaptation of a lower bound given in [22].

We also have demonstrated how our construction can be used to obtain dynamic embeddings of large binary trees with high load. The dilation of the embedding decreases to 6 while the load is nearly optimal. Again, this embedding can be constructed on the used hypercube efficiently. For a dynamic embedding with load ρ, the amortized time spent for each spawning step is $O(\rho \log(L/\rho))$ if in each step at most L new leaves are spawned.

Altogether, our algorithm presents an alternative way for embedding dynamically binary trees into hypercubes. Moreover, our construction has the advantage that simultaneous growing of new leaves is permitted. Furthermore, this is the first dynamic embedding of binary trees into hypercubes achieving constant load, dilation and node-congestion simultaneously.

It is easy to see that our dynamic embedding can be modified for arbitrary k-ary trees. It can be shown that the modified embedding achieves dilation of at most $O(\log(k))$ which is optimal up to a constant factor. Using an appropriate model of growing graphs, it turns out that our algorithm can also be extended to grow partial k-trees also known as graphs with bounded treewidth into hypercubes. In both cases, this extensions can be implemented efficiently on th hypercube.

References

[1] K. E. Batcher: Sorting Networks and Their Applications, *Proc. of the AFIPS Spring Joint Computer Conference*, **32**, 307–314, 1968.

[2] S. Bezrukov, B. Monien, W. Unger, G. Wechsung: Embedding Ladders and Caterpillars into the Hypercube, *Disc. Appl. Math.* **82**(1998), 19–27.

[3] S. Bhatt, J.-Y. Cai: Take a Walk, Grow a Tree, *Proc. of the 29th Symp. on Foundations of Computer Science*, 469-478, 1988.

[4] S. Bhatt, J.-Y. Cai: Taking Random Walks to Grow Trees in Hypercubes, *J. ACM*, **40**(1993), 741-764.

[5] S. Bhatt, F. Chung, T. Leighton, A. Rosenberg: Optimal Simulations of Tree Machines, *Proc. of the 27th Symp. on Foundations of Computer Science*, 274–282, 1986.

[6] S. Bhatt, F. Chung, T. Leighton, A. Rosenberg: Efficient Embeddings of Trees in Hypercubes, *SIAM J. Comput.*, **21**(1992), 151–162.

[7] S. Bhatt, I. Ipsen: How to Embed Trees in Hypercubes, *Yale University Research Report* RR-443, 1985.

[8] M.Y. Chan: Embedding of d-Dimensional Grids into Optimal Hypercubes, *Proc. of the 1989 Symp. on Parallel Algorithms and Architectures*, 52–57.

[9] M.Y. Chan: Embedding of Grids into Optimal Hypercubes, *SIAM J. Comput.*, **20**(1991), 834–864.

[10] M. Chan, F. Chin, C. Chu, W. Mak: Dilation-5 Embedding of 3-Dimensional Grids into Hypercubes, *J. Parallel Distrib. Comput.*, **33**(1996), 98-106.

[11] K. Efe: Embedding Mesh of Trees in the Hypercube, *J. Parallel Distrib. Comput.*, **11**(1991), 222–230.

[12] T. Feder, E. Mayr: An Efficient Algorithm for Embedding Complete Binary Trees in the Hypercube, *Stanford University*, 1987.

[13] C. Goldberg, D. West: Bisection of Circle Colorings, *SIAM J. Alg. Disc. Math.*, **6**(1985), 93–106.

[14] I. Havel: On Hamiltonian Circuits and Spanning Trees of Hypercubes (in Czech.), *Časopis. Pěst. Mat.*, **109**(1984), 145–152.

[15] I. Havel, P. Liebl: Embedding the Polytomic Tree into the n-Cube, *Časopis. Pěst. Mat.*, **98**(1973), 307–314.

[16] I. Havel, P. Liebl: One-Legged Caterpillars Span Hypercubes, *J. Graph Theory*, **10** (1986), 69–76.

[17] V. Heun, E. Mayr: A New Efficient Algorithm for Embedding an Arbitrary Binary Tree into Its Optimal Hypercube, *J. Algorithms*, **20**(1996), 375–199.

[18] V. Heun, E. Mayr: Embedding Graphs with Bounded Treewidth into Optimal Hypercubes, *Proc. of the 13th Symp. on Theoretical Aspects of Computer Science*, LNCS 1046, 157–168, 1996.

[19] V. Heun, E. Mayr: Optimal Dynamic Edge-Disjoint Embeddings of Complete Binary Trees into Hypercubes, *Proc. of the 4th Workshop on Parallel Systems and Algorithms*, 195-209, 1996.

[20] V. Heun, E. Mayr: A General Method for Efficient Embeddings of Graphs into Optimal Hypercubes, *Proc. of the 2nd International Euro-Par Conference on Parallel Processing*, Vol. I, LNCS 1123, 222-233, 1996.

[21] T. Leighton, M. Newman, A. Ranade, W. Schwabe: Dynamic Tree Embeddings in Butterflies and Hypercubes, *Proc. of the 1989 Symp. on Parallel Algorithms and Architectures*, 224–234.

[22] T. Leighton, M. Newman, A. Ranade, W. Schwabe: Dynamic Tree Embeddings in Butterflies and Hypercubes, *SIAM J. Comput.*, **21**(1992), 639–654.

[23] B. Monien, H. Sudborough: Simulating Binary Trees on Hypercubes, *Proc. of the 3rd Aegean Workshop on Computing*, LNCS 319, 170–180, 1988.

[24] D. Nassimi and S. Sahni: Parallel permutation and sorting algorithms and a new generalized connection network. *J. ACM*, 29(1982), 642–667.

[25] Y. Saad, M. Schulz: Topological Properties of the Hypercube, *Yale University Research Report* RR-389, 1985.

[26] J. Schwartz: Ultracomputers. *ACM Trans. Program. Lang. Syst.*, **2**(1980), 484–521.

[27] X. Sheen, Q. Hu, W. Liang: Embedding k-ary Complete Trees into Hypercubes, *J. Parallel Distrib. Comput.*, **24**(1995), 100–106.

[28] Q. Stout: Hypercubes and Pyramids, *Proc. of the NATO Advanced Research Workshop on Pyramidal Systems for Computer Vision 1986*, 75–89.

[29] A. Wu: Embedding of Tree Networks into Hypercubes, *J. Parallel Distrib. Comput.*, **2**(1985), 238-249.