IMFM
Institute of Mathematics, Physics and Mechanics
Jadranska 19, 1000 Ljubljana, Slovenia

Preprint series
Vol. 48 (2010), 1115
ISSN 2232-2094

EMBEDDING OF COMPLETE AND NEARLY COMPLETE BINARY TREES INTO HYPERCUBES

Aleksander Vesel

Ljubljana, March 12, 2010

Embedding of complete and nearly complete binary trees into hypercubes

Aleksander Vesel

Faculty of Natural Sciences and Mathematics, University of Maribor
Koroška cesta 160, 2000 Maribor, Slovenia
vesel@uni-mb.si

Abstract

A new simple algorithm for optimal embedding of complete binary trees into hypercubes as well as a node-by-node algorithm for embedding of nearly complete binary trees into hypercubes are presented.

Keywords: embedding; complete binary tree; hypercube; algorithm

1. Introduction and preliminaries

Hypercubes and binary trees are omnipresent in computer science. In particular, hypercubes are very popular model for parallel computation, because of their regularity, recursive structure and the ease of routing. On the other hand, the binary tree can represent the basic computational structure of divide-and conquer or branch-and-bound algorithms. In many cases however, it is more suitable that the internal structure of an algorithm is modeled by a more general structure - a nearly complete binary tree.

In this paper we consider the problem of embedding the (nearly) complete binary tree in the hypercube. This problem develops in the implementation of divide-and conquer algorithms in a hypercube network, e.g. see [2, 4]. An embedding is a mapping from the guest graph, representing the communication structure of the processes, into the the host graph, representing the communication network of the processors. Therefore, the problem of allocating processes to processors in a multiprocessor system is also known as the *mapping problem*.

A *tree* is a connected acyclic graph. One vertex is distinguished and called the *root*. A vertex of degree one is called a *leaf* of the tree if it is not the root. The *level* of a vertex v in a tree is the number of vertices on the simple path from the root to v. Note that the level of the root is one. The height of a tree T is the largest level of a vertex in T. A vertex u is called a *child* of v if u is adjacent to v and the level of u is bigger then the level of v. If u is a child of v then v is called the *parent* of u.

A *full binary tree* is a tree in which every node other than the leaves has two children. A full binary tree is ordered, i.e. we distinguish between left and right children. A *complete binary tree* is a full binary tree in which all leaves are at the same level. A *nearly complete binary tree* of height h is composed of a complete binary tree of height $h-1$ and with some nodes at level h.

The *hypercube* of order d and denoted Q_d is the graph $G = (V, E)$ where the vertex set $V(G)$ is the

set of all binary strings $u^{(1)}u^{(2)}\dots u^{(d)}$, $u^{(i)} \in \{0,1\}$. Two vertices $x, y \in V(G)$ are adjacent in Q_d if and only if x and y differ in precisely one place.

An *isomorphic embedding* (or just an *embedding*) of a graph $G(V,E)$ into a graph $H(V',E')$ is an injection $f : V \to V'$ such that if (u,v) is an edge in $E(G)$ then $(f(u), f(v))$ is an edge in $E'(H)$.

For binary vectors $s, t \in \{0,1\}^n$ let $s \oplus t$ denote the coordinate wise addition modulo two, e.g. $100011 \oplus 000001 = 100010$. Let e_i^n be the binary vector $(u_1, u_2, \dots, u_n)$ with $x_i = 1$ and $x_j = 0$, $j \neq i$. If s is a binary vector of length n, then we will call the operation $e_i^n \oplus s$ a *reflection*. We will also use the "+" symbol as the concatenation operator, i.e. $s + t$ joins two binary vectors s and t end to end. If we concatenate a binary vector with a single bit (0 or 1), than we call this operation a *projection*. For a binary vector s of length n, $s + 0$ and $s + 1$ are projections of s into two disjoint hypercubes of order n which compose Q_{n+1}.

It is a natural question to ask for an embedding into a hypercube with the least order. The minimum h required for an embedding from a graph G into Q_h is called the *cubical dimension* of G. Deciding whether there exists an embedding of a given tree into a hypercube of a given dimension is known to be NP-complete [3]. Moreover, even in case of trees with bounded degrees, their cubical dimensions are unknown in most cases.

Obviously, if G is a graph such that $2^h \geq |V(G)| > 2^{h-1}$, then the cubical dimension of G is at least h. However, it is well known that the complete binary tree on $2^h - 1$ vertices cannot be embedded into Q_h, i.e. into its "optimal" hypercube.

Havel in 1984 conjectured that every binary tree T with $2^h \geq |V(T)| > 2^{h-1}$ vertices has an embedding into Q_{h+1}. The conjecture is still open, but there are many partial results supporting this assertion. It has been showed, for example, that a complete binary tree of height h can be embedded into the hypercube of order $h + 1$, e.g. [2, 4].

This paper present a new simple algorithm which embeds a complete binary tree of height h into the hypercube of order $h + 1$. This algorithm is presented in Section 2. Moreover, the algorithm is the basis for the embedding of nearly complete binary trees of height h into Q_{h+1}, which is presented in Section 3.

2. Complete binary tree

Let C_h denote the complete binary tree of height h.

Let also r_h denote the root of C_h and let C_h^l and C_h^r denote the left and the right subtree of C_h, respectively. Obviously, C_h^l and C_h^r are both complete binary trees of height $h - 1$.

Let also define the mapping $\sigma_l : V(C_h^l) \to V(C_{h-1})$, where for a vertex $v \in C_h^l$ its image $\sigma_l(v)$ denote the corresponding vertex of C_{h-1} in a natural way, e.g. if v is the root of C_h^l, then $\sigma_l(v) = r_{h-1}$. Analogously we define the mapping $\sigma_r : V(C_r^l) \to V(C_{h-1})$.

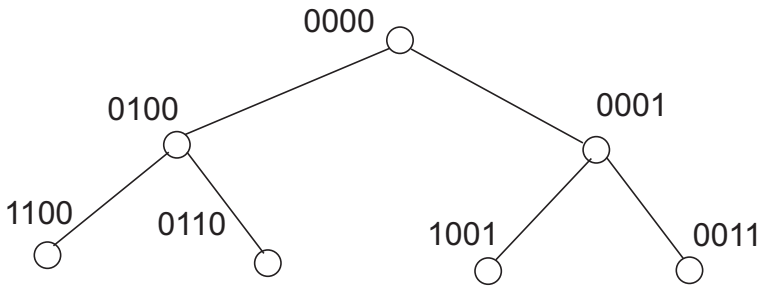

Figure 1: β_3 - the mapping of C_3

Let β_h be a mapping that determines a binary string of length $h+1$ to every vertex of the complete binary tree of height $h \geq 3$. Formally, $\beta_h : V(C_h) \to \{0,1\}^{h+1}$ is the mapping, such that β_3 maps the vertices of C_3 as depicted in Fig. 1, while for $h > 3$ the mapping is given by

$$\beta_h(v) = \begin{cases} 0^{h+1}, & v = r_h \\ \beta_{h-1}(\sigma_r(v)) + 1, & v \in V(C_h^r) \\ \beta_{h-1}(\sigma_l(v)) \oplus 10^{h-1} + 0, & v \in V(C_h^l) \text{ and } h \text{ is even} \\ \beta_{h-1}(\sigma_l(v)) \oplus 0^{h-3}100 + 0, & v \in V(C_h^l) \text{ and } h \text{ is odd}\,. \end{cases}$$

For any v of C_h, the string $\beta_h(v)$ will be also called a *codeword* defined by β_h in the sequel.

Theorem 1. *Let $h \geq 3$. Then β_h makes an embedding of the complete binary tree of height h in the hypercube of order $h+1$.*

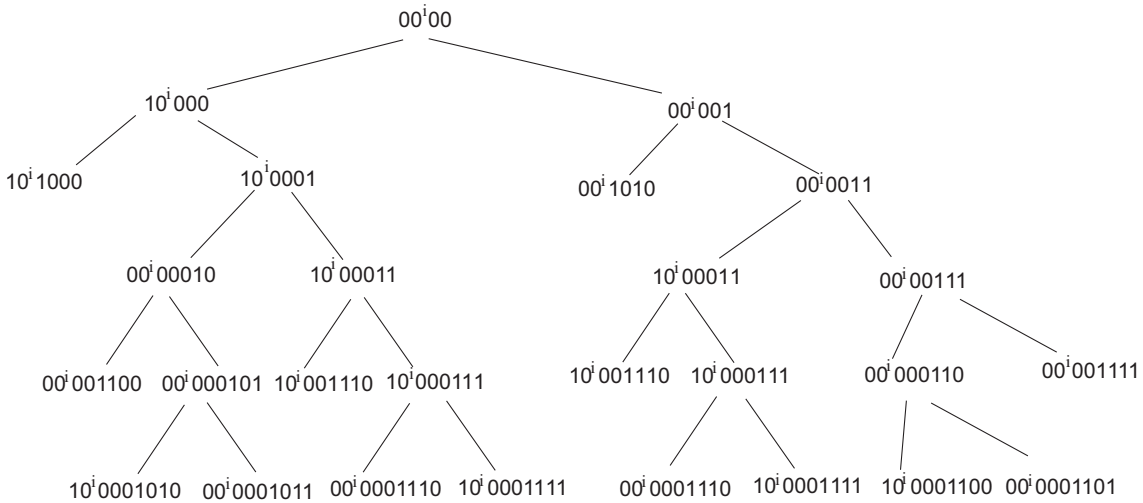

Figure 2: Codewords derived from 00^i0 when i is even

The basis of the proof is the following lemma.

Lemma 1. *Let $h \geq 3$. Then $\beta_h(v) = 0^{h+1}$ if and only if v is the root of C_h.*

Proof. If v is the root of C_h, then by the definition $\beta_h(v) = 0^{h+1}$. Therefore we only have to show that if v is not the root of C_h, then $\beta_h(v) \neq 0^{h+1}$. Note first, that the recursive definition of β_h implies, that if

v is not the root of C_h, then any $\beta_h(v)$ is derived as a sequence of projections and reflections either from the root of the C_{i+1}, i.e. $\beta_{i+1}(r_{i+1}) = 00^i0$, $h > i \geq 3$, or from a codeword defined by β_3.

Suppose first that $\beta_h(v)$ derives from 00^i0. Suppose also that i is even. Some of the codewords derived from 00^i0 can be seen in the tree depicted in Fig. 2. The root of the tree is 00^i0, while the left and the right child of 00^i0 is obtained as a codeword derived from it in the left and the right subtree of C_{i+2}, respectively. Analogously, the left and the right children of 10^i000 and 00^i001 are codewords in C_{i+3}, etc.

Note that the definition of β_j for $j \geq 4$ implies that $\beta_j(v)$ is derived from a codeword s of C_{j-1} such that either the first or the $(j-3)$-th bit of s is reversed. It follows that from a codeword s of C_{j-1} with 1 on at least one of the places: $2, 3, \ldots, j-4$, a codeword of the form 0^{h+1} cannot be derived. Since every leaf of the tree from Fig. 2 in at least one of those places possesses entry 1, it follows that 0^{h+1} cannot derive from 00^i0 if i is even. It is not difficult to see that a similar tree (having leaves of length j with 1 on at least one of the places $2, 3, \ldots, j-4$) can be derived for 00^i0, where i is odd.

For $\beta_h(v)$, which derives from a codeword defined by β_3, observe first that all codewords of the left subtree of C_3 depicted in Fig. 1 posses 1 at the second place, which implies that 0^{h+1} cannot be derived from any of them. For $\beta_h(v)$ derived from a codeword of the left subtree of β_3, observe the trees depicted in Fig. 3. From the same arguments as above we now conclude that 0^{h+1} cannot derive from any codeword defined by β_3 and the proof is complete.

□

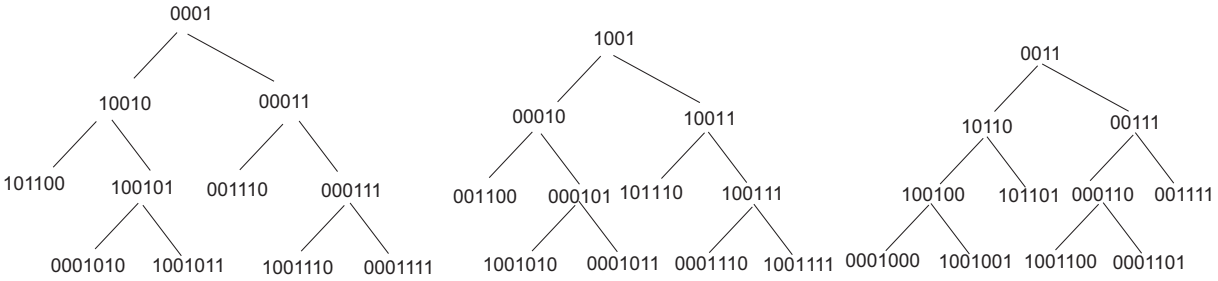

Figure 3: Some codewords derived from β_3

Proof (of Theorem 1). The proof is by induction on h. The claim obviously holds for β_3. Suppose also that the claim holds for h. Note first that projections maps the vertices of C^l_{h+1} and C^r_{h+1} into two disjoint hypercubes. Moreover, reflections of an embedding in a hypercube preserve Hamming distance. Therefore β_{h+1} is an embedding of C^l_{h+1} and C^r_{h+1} in the hypercube of order $h+1$. Finally, since the root of C_{h+1} and the root of C^l_{h+1} (as well as C^r_{h+1}) differ in precisely one bit, we conclude that β_{h+1} embeds C_{h+1} into Q_{h+1} and the proof is complete. □

Theorem 1 is the basis for the algorithm to compute the optimal embedding of the complete binary tree of height h in Q_{h+1}.

We first present the algorithm to calculate the embedding of the complete binary tree of height h from the embedding of the complete binary tree of height $h-1$. It is assumed in the algorithm, that if v is an arbitrary node of a complete binary tree T, then $b(v)$ is the codeword of v and $p(v)$ a parent of v in T. Furthermore, r denote the root of T and T_l and T_d denote the left and the right subtree of T, respectively.

Procedure NEW TREE

input: h, T, b {T is a complete binary tree of height $h > 3$, b is the embedding of the complete binary tree of height $h-1$ }

output: b { The embedding of the complete binary tree of height h }

begin

traverse T_r from level h to level 2 and for every $v \in T_r$ **do**

$b(v) := b(p(v))) + 1$;

if $h \bmod 2 = 0$ **then**

traverse T_l from level h to level 2 and for every $v \in T_l$ **do**

$b(v) := b(p(v)) \oplus 10^{h-1} + 0$;

else

traverse T_l from level h to level 2 and for every $v \in T_l$ **do**

$b(v) := b(p(v))) \oplus 0^{h-3}100 + 0$;

$r := 0^{h+1}$; { The new root of T }

end.

We next describe the algorithm to compute the optimal embedding of the complete binary tree of height h in a hypercube.

Procedure CODES

input: h {height of a tree, $h \geq 3$}

output: T, b { T is a complete binary tree of height h with the embedding b }

begin

1. $T :=$ complete binary tree of height 3.
2. Determine $b(v)$ for every node of $v \in T$ as in Fig. 1;
3. **for** $i := 4$ to h **do begin**

Augment T with new level of nodes to obtain the complete binary tree of height i;

NEW TREE(i, T, b);

end.

Theorem 2. *Let $h \geq 3$. Then CODES embeds a complete binary tree of height h into Q_{h+1} in linear time and space.*

Proof. The correctness of the algorithm is by induction on h. If $h = 3$, then the embedding is given in Step 2, the correctness of which can be verified by Fig. 1.

Assume now that for $i = h - 1$ the algorithm correctly compute the embedding of T. In other words, when NEW TREE is called in Step 3 for $i = h$, the vector b corresponds to the embedding β_{h-1}. Moreover, for a node $v \in T_l$ ($v \in T_r$), the old value of $b(p(v))$ corresponds to $\beta_{h-1}(\sigma_r(v))$ ($\beta_{h-1}(\sigma_l(v))$). Therefore, since the nodes of T are traversed from the last level to the roots of the subtrees and since NEW TREE accurately follows the definition of β_h, we can conclude that the embedding is correct.

T, p, and b can be obviously represented in linear space, therefore we only consider the time complexity. Let then $n = 2^h - 1$ denote the number of nodes of a complete binary tree of height h. Note first that NEW TREE computes the embedding b in time which is linear in the size of T. Since the number of vertices of T in i-th iteration of the **for** loop equals $2^i - 1$, the total number of steps of the algorithm is given by

$$\sum_{i=4}^{h} 2^i - 1 = 2^{h+1} - 12 = O(n) \, .$$

This argument completes the proof.

□

3. Near complete binary tree

In this section we present a simple node-by-node algorithm for constructing an embedding of a nearly complete binary tree into a hypercube. We assume that in each time step a nearly complete binary tree can grow for one node, which is inserted at the last level of a tree. Note, that in [1] a somewhat similar approach has been studied, where the complete binary tree grows by a complete level of its leaves.

The algorithm presented herein compute the map of the nodes of a new tree using the map of their parent node. Moreover, if a new node does not change the height of a tree, the old nodes need not to be remapped.

The algorithm of the previous section implies, that the embedding of a complete complete binary tree of height h can be performed by using the embedding of the complete complete binary tree of height $h - 1$, such that the embedding of a node v is computed from the "old" embedding of its parent node.

This observation leads to a node-by-node algorithm for embedding of nearly complete binary trees into hypercubes. The algorithm augments a given nearly complete binary tree with one node, which is inserted at the level h and computes the mapping of the augmented tree. We will show that in the majority of cases the algorithm is able to determine the embedding of the augmented tree by simply expanding the embedding with the map of the new node. Moreover, the map of the new node can be computed with ease from the map of its parent node.

If a nearly complete binary tree T of height h with embedding into Q_{h+1} is augmented with one new node, then for the resulting nearly complete binary tree T' the embedding into Q_{h+1} (or Q_{h+2}, if the height of T' is $h+1$) is computed. The new node can be

(i) a leaf at level h, if T is not complete or

(ii) a leaf at level $h+1$, if T is complete.

Let β_h be a mapping that determines a binary string of length $h+1$ to every vertex of the complete binary tree of height $h \geq 3$ as defined in Section 2. In order to obtain the embedding for T' we first show the following lemma.

Lemma 2. *Let for $h \geq 3$, v be a leaf of C_h and u the parent of v in C_h. Then*

$$\beta_h(v) = \begin{cases} \beta_h(u) \oplus 10^h, & v \text{ is the left child of } u \\ \beta_h(u) \oplus 001^{h-3}0, & v \text{ is the right child of } u. \end{cases}$$

Proof. The proof is by induction with respect to h. The claim obviously holds if $h = 3$ as can be seen in Fig 1. Let us denote v a leaf of C_{h+1} and u the parent of v. If v (and u) is in the right subtree of C_{h+1}, then by inductive hypothesis $\beta_h(\sigma_r(v))$ and $\beta_h(\sigma_r(u))$ differ either in the first bit, if v is the left child of u, or in the third bit, if v is the right child of u. It is straightforward to see now that $\beta_{h+1}(v) = \beta_h(\sigma_r(v))+1$ and $\beta_{h+1}(u) = \beta_h(\sigma_r(u)) + 1$ differ either in the first or in the third bit.

If v is in the left subtree of C_{h+1}, the proof is analogous. □

In the following algorithm, let for an arbitrary vertex u of a nearly complete binary tree T, $b(u)$ and $p(u)$ denote the codeword of u and the parent of u in T, respectively.

Procedure NEW NODE

input: h, T, b, v { b is the embedding of T, $h \geq 3$, v is a new node }

output: T, h, b { An augmented tree of height h with the embedding b }

begin

1. **if** T is complete **then begin**

   NEW TREE (h, T, b);

$h := h + 1$;

end;

2. Insert v at the level h in T;

3. **if** v is the left child of $p(v)$ **then**

$b(v) := b(p(v))) \oplus 10^h$;

else $b(v) := b(p(v))) \oplus 0010^{h-2}$;

end.

In order to obtain the embedding of a nearly complete binary tree with NEW NODE, before the algorithms is first called, Step 1 and Step 2 of CODES have to be executed. T is then the complete binary tree of height 3 with the embedding b.

Theorem 3. *If T is C_3 or a nearly complete binary tree of height $h > 3$ and b the embedding of T into Q_{h+1}, then NEW NODE correctly embeds T' either*

(i) into Q_{h+2}, if T is complete or

(ii) into Q_{h+1}, if T is not complete.

Proof. Assume that T is either C_3 or an arbitrary near complete binary tree of height $h > 3$ and b its embedding into Q_{h+1}. If T is not complete, then the correctness of the algorithm follows from Lemma 2.

Let then T be a complete binary tree. When NEW TREE is called in Step 1, the value of h is not yet incremented, i.e. the output of the procedure is the complete tree of height h with the embedding into Q_{h+2}. However, in Steps 2 and 3, T is first augmented with a new node at level $h + 1$ and then the embedding into Q_{h+2} of the resulting nearly complete tree of height $h + 1$ is computed.

□

The following concluding comment concerning the time complexity of the algorithm is on order. The algorithm remaps the nodes of T only if T is a complete binary tree. In other cases a remapping is not performed. Clearly, the embedding of a new node depends only on the map of its parent node and can be performed in constant time. However, even in the case when remapping is needed, the computation of the new embedding can be done independently in each node v such that only the codeword of a parent node of v is used. It follows that the remapping can be computed on the hypercube in parallel in constant time.

References

[1] V. Heun and E.W. Mayr, Optimal dynamic embeddings of complete binary trees into hypercubes, *J. Parallel Distrib. Comput.* **6** (2001) 11110–1125.

[2] A.S. Wagner, Embedding the complete tree in the hypercube, *J. Parallel Distrib. Comput.* **20** (1994) 241–247.

[3] A.S. Wagner, D.G. Corneil, Embedding trees in a hypercube is NP-complete, *SIAM J. Comput.* **7** (1990) 570–590.

[4] A. Y. Wu, Embedding of Tree Networks into Hypercubes, *J. Parallel Distrib. Comput.* **2** (1985) 238–249.