Asymptotic Hypercube Embeddings of Dynamic k-ary Trees

Ömer Eğecioğlu [*] and *Maximilian Ibel*

Department of Computer Science
University of California Santa Barbara, CA 93106
{omer,ibel}@cs.ucsb.edu

Abstract

Several algorithms are known for embedding dynamically growing trees onto hypercubes. In a dynamic k-ary tree each leaf node may spawn k new children at any given time. The embedding process must not reassign any tree node to another host node in the hypercube once it has been placed. Desirable properties of the embedding are low dilation and optimal load-balance.

The existing algorithms are mainly directed toward optimizing the load balance for trees that are comparable in size to the host graph. It has been observed that in this case the naïve approach of assigning newly spawned leaves to random neighbors in the hypercube host yields suboptimal results. We consider the asymptotic behavior of this naïve placement algorithm. For symmetry reasons it is to be expected that the resulting process should lead to an asymptotically balanced load for dynamic k-ary trees. We give a formal proof of this based on the Matrix-tree theorem for graphs. The proof generalizes to arbitrary connected regular host graphs, such as tori networks.

Keywords: Embedding, hypercube, dynamically growing trees, load balancing, random walk, Matrix-tree theorem.

1 Introduction

Often the structure of computations is not fixed beforehand (as, for example, for the Fast Fourier Transform), but rather depends on the input data or other run time variables. Examples for such irregular computations are branch-and-bound methods, where the tasks form some adaptive tree

[*] Supported in part by NSF/NASA/ARPA Grant No. IRI94-11330.

and the children of a task are sub-tasks that have been forked. Also, some adaptive grid methods can be described as quad-trees that grow dynamically whenever the grid is refined.

Many of these irregular computations are so time-consuming that they are solved on parallel machines. Naturally the question arises how to distribute (or embed) the dynamic set of tasks to the hardware, which is modeled as a graph whose nodes correspond to the individual computers and edges to the network links. The topology of common multicomputers are hypercubes (e.g., the Intel NCUBE), meshes (e.g. MasPar MP-1), tori (e.g. the Cray T3E), rings (e.g., Kendall Square's KSR-1), or topologies easily embeddable into the hypercube (e.g., BBN Butterfly). We refer the reader to Leighton [Lei92] for a detailed description and properties of these architectures.

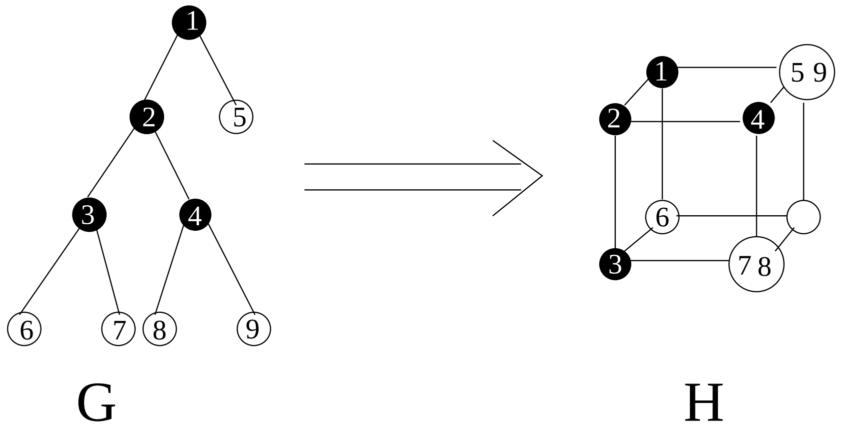

Figure 1: Embedding of a binary tree G with 5 leaves into a three-dimensional hypercube H with dilation 1 and load 2.

Some properties a good embedding must have are:

1. **Practicality.** It must be straightforward to place a newly generated task on a host node. In particular, it is not feasible to inquire the status of all host nodes to determine the least loaded host node close to the host node of the parent tree node.

2. **Low dilation.** A child task generated from a parent task should not be placed on a host node far away from the host node of the parent.

3. **Good load balance.** All host nodes should at all times have approximately the same number of tasks. If we assume all tasks take the same time, and communication occurs only between parents and

children, then nodes which receive a higher proportion of tasks will take longer than other nodes, and the total computation time will be longer.

The two seminal publications on dynamic embedding of trees into hypercubes are due to Bhatt and Cai [BC89], and Bhatt, Chung, Leighton, and Rosenberg [BCLR92]. Bhatt and Cai gave an embedding of an M-node binary tree into an N-node hypercube, where the children of a tree node are determined by a random walk of length $\log \log N$ in the hypercube. These embeddings have load $O(M/N+1)$ and dilation $O(\log \log N)$. In [BCLR92] an algorithm based on a specialized random walk of length 1 with dilation 1 and load $O(M/N + \log N)$ is presented. Both algorithm are targeted to yield good results for small M (e.g., $M \approx N$). A more recent treatise of dynamical tree embeddings is given in [HM96]. Our contribution is an analysis of an even simpler embedding algorithm for the case relevant in practial applications, that is when $M \gg N$. This *naïve placement algorithm* turns out to yield asymptotically perfect load balancing. We show that this follows from the Matrix-tree theorem.

As in [BC89, BCLR92], we assume that once a task has been assigned to a particular host node, it stays there until its completion. Thus we do not allow for task migration. Furthermore, we do not deal with deletion of nodes and consider growing trees only, as the reduction of the case of mixed insertions and deletions to insertions only is discussed in [BC89] and [BCLR92]. Our analysis is valid not only for hypercubes but also for arbitrary connected processor graphs which are regular, such as tori networks.

The paper is organized as follows. Section 2 introduces the formal problem description and terminology used throughout the paper. Section 3 describes three different algorithms used to embed dynamically evolving trees into hypercubes, Section 4 shows that the naïve placement algorithm asymptotically achieves optimal load balance. Section 5 shows some simulation results for the three algorithms. Section 6 concludes the paper and offers future research directions.

2 Notation and Problem Formulation

In the following, we consider the embedding of a process graph G (guest graph) into a processor graph H (host graph). The nodes V_G in $G = (V_G, E_G)$ represent the tasks, and the edges E_G the dependencies (we assume only depending tasks need to exchange information), whereas nodes and edges in $H = (V_H, E_H)$ represent processors and communication links, respectively. Let us denote the number of tasks $|V_G|$ by M and the number of processors $|V_H|$ by N.

Formally, an embedding is defined as a node-mapping $f : V_G \to V_H$ and an edge-mapping $g : E_G \to \mathcal{P}(H)$, where $\mathcal{P}(H)$ is the path set of H and the *routing path* $g(\{u, v\})$ connects $f(u)$ and $f(v)$ in V_H, for $\{u, v\} \in E_G$.

The following metrics are commonly associated with the embedding problem: The *dilation* of the embedding (f, g) is the maximum length of the routing-path connecting two adjacent nodes in the guest graph. The *load* is the maximum, over all host-nodes, of the number of guest-nodes mapped to a host-node. The *edge congestion* is the maximum number of edges of G that are routed by the mapping g over a single edge of H.

Ideally, the load of each host node is M/N (i.e., the work is shared equally), and the dilation is 1 (i.e., adjacent nodes in G are mapped on adjacent nodes in H). It turns out that load and dilation are conflicting parameters: If the dilation is kept small, we need to increase the maximum load, whereas good load balancing is facilitated by allowing for higher dilation. In order to simplify our analysis, we assume that each task takes unit time to execute on any processor, and only leaf nodes are contributing to the computation at any time. Therefore, the load of any host node is the number of tree leaves mapped to it.

We model dynamic k-ary trees as follows: starting with a single root node, we repeatedly generate larger trees. In each step, we choose one of the leaves of the current tree. This leaf then spawns k new children. A example for $k = 2$ is given in Figure 2, where the three possible successors of a binary tree with a total of 3 leaves are shown. We assume that the leaf picked for expansion in each step is chosen with equal probability from the set of all current leaves of the tree.

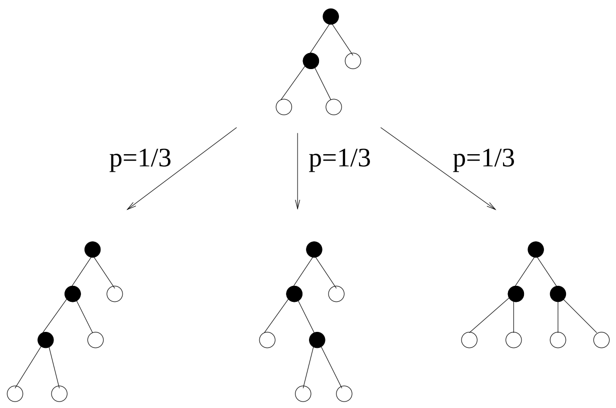

Figure 2: Three possible successors generated with equal probability 1/3 from a binary tree with 3 leaves.

Let H_d denote the d-dimensional hypercube, whose nodes are numbered from 0 to $N-1$, where $N = 2^d$. The $d = \log N$ nodes adjacent to x are given by $x[i] = x$ XOR 2^i, for $0 \le i < d$. We also use the notation $x \leftrightarrow y$ for adjacent nodes x and y.

3 Three Tree Embedding Algorithms

3.1 Naïve Placement Algorithm

The most straightforward embedding algorithm, henceforth called *naïve placement*, starts by mapping the root of the source tree G to a fixed node, say node 0 of the host hypercube H. When a leaf node u which has already been mapped to a host node x in the hypercube is expanded, we randomly choose k of the d neighbors of x to assign the newly spawned k children of u. We analyze this algorithm in Section 4.

3.2 Random Walk Algorithm

It has been noted that the naïve placement does not perform well for some task trees of approximately the same size as the host graph, for example complete binary trees. Bhatt and Cai [BC89] proposed the following embedding algorithm: A newly spawned child v is not placed at a neighbor of the hypercube node x hosting the parent u of v, but rather, a random walk of length $O(\log\log N)$ starting from x is taken, and the new host node is taken to be the endpoint of this walk. Bhatt and Cai showed that with high probability, the resulting load is $O(M/N + 1)$, and the dilation is $O(\log\log N)$. They also showed that if the random walk length is chosen smaller, say $o(\log\log N)$, and the embedded tree is a complete binary tree with $N \approx M$, then at least one node has load $\Omega(\log^k N)$ for any k, implying that the load balance will be very poor in this case.

3.3 Flip-Bit Algorithm

The dilation of $O(\log\log N)$ is a high price to pay for acceptable load balancing. Bhatt et al. [BCLR92] showed how to embed binary trees into butterfly networks with load $O(M/N + \log N)$ and dilation 1 with high probability. The *flip-bit* algorithm can also be generalized to hypercubes, yielding the same load and dilation. Each node in the tree is assigned a dimension number equal to its depth in the tree modulo $\log N$. When a leaf with dimension number n spawns a child, it either is mapped to the parent's host node or the node reachable from the parent host node by inverting bit n. The same paper also describes an improved algorithm that achieves load $O(M/N+1)$ and dilation $O(1)$. The constant for the dilation for this algorithm is 12.

The main reason for the rejection of the naïve placement algorithm was its unsuitability for trees equal in size to the host hypercube. We therefore analyze in the next section the performance of the naïve placement algorithm for large M/N, as N depends on the hardware and stays fixed while M can get arbitrarily large.

4 Asymptotic Behavior of the Naïve Placement Algorithm

We first show how the Matrix-tree theorem implies that there exists exactly one load distribution that is stationary with respect to the expansion of nodes. A formal proof of convergence follows by modeling the embedding process as a non-stationary Markov chain with convergent time-dependent transition probabilities.

Define the *load vector* $L = (L_0, L_1, \ldots, L_{N-1})$, where L_i is the number of leaves hosted by hypercube node i. $M = L_0 + \cdots + L_{N-1}$ is the total number of leaves. Initially, the tree consists only of a root node hosted at processor 0, so that $L = (1, 0, 0, \ldots, 0)$. Clearly, for load-balance we want all elements in the load vector to be of the approximately the same size.

As an example for $k = 2$, let us consider the 3 dimensional hypercube with nodes 0 to 7, and binary guest trees. Table 1 shows possible mapping of children to hypercube nodes for all possible parent nodes. For example, if a leaf hosted by hypercube node 0 is expanded the two new leaf news will be placed at $\{1,2\}$, $\{1,4\}$, or $\{2,4\}$.

Parent	Choice 1	Choice 2	Choice 3
0	$\{1,2\}$	$\{1,4\}$	$\{2,4\}$
1	$\{0,2\}$	$\{0,3\}$	$\{2,3\}$
2	$\{0,3\}$	$\{0,5\}$	$\{3,5\}$
3	$\{1,2\}$	$\{1,7\}$	$\{2,7\}$
4	$\{0,5\}$	$\{0,6\}$	$\{5,6\}$
5	$\{1,4\}$	$\{1,7\}$	$\{4,7\}$
6	$\{2,4\}$	$\{2,7\}$	$\{4,7\}$
7	$\{3,5\}$	$\{3,6\}$	$\{5,6\}$

Table 1: Possible pairs of children mappings for each parent in the hypercube.

In general, consider the d-dimensional hypercube ($d = \log N$), in which a leaf spawns k new children. There are $\binom{d}{k}$ choices for placing the newly created leaves[1]. Once this placement is made, let us consider the change

[1] k is taken to be (k *mod* d) if $k > d$.

δ_L on the current load vector. Since we are counting leaves only towards the load, the expanded node will vanish from the load vector and the corresponding entry in the load vector is decremented. Similarly, the entries in the load vector corresponding to the k newly placed children will be incremented.

Let us again consider our example and assume a leaf on hypercube node 0 is expanded. Then, the 3 choices to place 2 children, $\{1,2\}$, $\{1,4\}$, and $\{2,4\}$, will result in the relative changes $\delta_L = (-1,1,1,0,0,0,0,0)$, $\delta_L = (-1,1,0,0,1,0,0,0)$, and $\delta_L = (-1,0,1,0,1,0,0,0)$ to the load vector, respectively.

Given a particular load vector L, the probability p_i of choosing node i for expansion is $\frac{L_i}{L_0+\ldots+L_{N-1}} = \frac{L_i}{M}$. Now consider the probabilities by which entries in the load vector change:

1. The probability L_i is decremented is p_i, since L_i is decremented if and only if node i is expanded.
2. The probability that L_i is incremented is $\frac{k}{d}\sum_{j\leftrightarrow i} p_j$. The factor $\frac{k}{d}$ is obtained as follows. Since each neighbor j picks k of the d neighbors, we need to calculate the probability that node i is chosen as one of the children. This probability is $\binom{d-1}{k-1}/\binom{d}{k} = \frac{k}{d}$.
3. The probability L_i remains unchanged is $q_i = 1 - p_i - \frac{k}{d}\sum_{j\leftrightarrow i} p_j$, since this event is complementary to the first two.

In our example, we find

1. $P(L_0 \text{ is decremented }) = p_0$,
2. $P(L_0 \text{ is incremented }) = \frac{2}{3}(p_1 + p_2 + p_4)$,
3. $P(L_0 \text{ is unchanged }) = q_0 = 1 - p_0 - \frac{2}{3}(p_1 + p_2 + p_4)$.

For a stationary load vector (i.e. at equilibrium), the ratio of leaves allocated to each host-node to the total number of nodes remains relatively unchanged after expanding nodes, while the total number of nodes increases from M to $M + k - 1$. We therefore obtain

$$p_i \frac{(L_i - 1)}{M + k - 1} + \frac{k}{d}(\sum_{j\leftrightarrow i} p_j)\frac{(L_i + 1)}{M + k - 1} + q_i \frac{L_i}{M + k - 1} = \frac{L_i}{M}.$$

This means

$$\frac{L_i - p_i + \frac{k}{d}\sum_{j\leftrightarrow i} p_j}{M + k - 1} = p_i \ . \qquad (1)$$

Since for positive x, y, a, b, r the implication $\frac{x}{y} = r \wedge \frac{x+a}{y+b} = r \Rightarrow \frac{a}{b} = r$ holds, we obtain from (1)

$$\frac{-p_i + \frac{k}{d}\sum_{j\leftrightarrow i} p_j}{k-1} = p_i,$$

or after rearranging

$$dp_i - \sum_{j\leftrightarrow i} p_j = 0.$$

Using the last equation for all nodes i in the host graph, we obtain the matrix equation

$$(\Delta - \mathbf{A})(p_0, p_1, \dots, p_{N-1})^T = \mathbf{0}. \tag{2}$$

Here, Δ is the diagonal matrix with d's down the diagonal, and $\mathbf{A}$ the adjacency matrix of the hypercube. The matrix $\Delta - \mathbf{A}$ is the so-called *Laplacian* of H. The Matrix-tree theorem states that every cofactor of the Laplacian is equal to the number of spanning trees of H, a positive integer. Therefore the determinant of every $(N-1)\times(N-1)$ submatrix is nonzero, and hence the subspace of solutions of (2) is at most one-dimensional. Since we can easily verify that $p_0 = p_1 = \cdots = p_{N-1}$ is a solution, the solution space is exactly one-dimensional. Therefore the stationary probabilities are $p_0 = p_1 = \cdots = p_{N-1} = \frac{1}{N}$.

Next, we go on to prove that the embedding process will yield a limiting distribution of tree nodes onto hypercube nodes.

Instead of the equation for a stationary distribution, we now have a recursion formula. Let $L(t) = (L_0(t), \dots, L_{N-1}(t))$ denote the load vector at time t, $M(t)$ denote the number of leaves at time t, and $x_i(t) = L_i(t)/M(t)$ denote the expected load at node i at time t. Since in each expansion step the number of leaves grows by $k-1$, $M(t) = t(k-1)+1$. Therefore

$$L_i(t) = \frac{L_i(t-1)}{M(t-1)}(L_i(t-1)-1) + \frac{k}{d}\sum_{i\leftrightarrow j}\frac{L_j(t-1)}{M(t-1)}(L_i(t-1)+1) +$$

$$+\left(1 - \frac{L_i(t-1)}{M(t-1)} - \frac{k}{d}\sum_{i\leftrightarrow j}\frac{L_j(t-1)}{M(t-1)}\right)L_i(t-1),$$

or in terms of the expectations $x_i(t)$:

$$x_i(t) = x_i(t-1)\frac{M(t-1)}{M(t)} + \frac{k}{d}\sum_{i\leftrightarrow j}\frac{x_j(t-1)}{M(t)} - \frac{x_i(t-1)}{M(t)}. \tag{3}$$

If we write the latter equation in matrix form, denoting by $X(t)$ the vector $(x_0(t), \ldots, x_{N-1}(t))^T$, we obtain

$$X(t) = \mathbf{C_t} X(t-1)$$

where

$$\mathbf{C_t} = \left(\frac{M(t-1)-1}{M(t)}\right)\mathbf{I} + \frac{k}{dM(t)}\mathbf{A}.$$

We only need to prove that $\lim_{t\to\infty} \mathbf{C_t} \cdot \mathbf{C_{t-1}} \cdots \mathbf{C_1}$ exists. The application of the Matrix-tree theorem then shows that $X(t) \to (\frac{1}{N}, \frac{1}{N}, \ldots, \frac{1}{N})^T$. We only give a sketch of the proof of the existence of this limit. It is known that for a d-regular graph, d is an eigenvalue of maximal modulus of the adjacency matrix $\mathbf{A}$ [Big93]. Let $\mathbf{T}$ be the matrix that brings $\mathbf{A}$ to its Jordan canonical form $\mathbf{F}$: $\mathbf{F} = \mathbf{TAT^{-1}}$. Let $\mathbf{B_t} = \mathbf{TC_tT^{-1}}$. Then

$$\mathbf{B_t} = \begin{pmatrix} \theta_0 & * & & & \\ & \theta_1 & * & & \\ & & \ddots & & \\ & & & \theta_{N-2} & * \\ & & & & \theta_{N-1} \end{pmatrix}, \tag{4}$$

where $|\theta_i| \leq 1$ and the elements denoted as $*$ are $O(1/t)$. An estimate for the growth rate of the entries in the product then shows that $\mathbf{B_t} \cdot \mathbf{B_{t-1}} \cdots \mathbf{B_1}$ converges.

It can be shown using the recursion (3) that the variance of the x_i goes to 0, so that the convergence to equal load is not only in expectation, but also in probability.

We conclude this section with two important observations: The key part of our analysis was the proof that equal load is the unique stationary distribution. Therefore, we can replace the hypercube as host graph with an arbitrary connected d-regular graph, and the same argument goes through. Secondly, we used a randomized method to generate k-ary trees. However, the embedding algorithm and our analysis does not depend on the shape of the guest tree generated. In particular, we can replace random trees by complete binary trees and still obtain good load distribution.

5 Experimental Results

We implemented the three embedding algorithms on two data sets. First, we generated random binary trees as described in Section 2 with a total of $2^{19} - 1$ nodes (2^{18} leaves). We then embedded the trees into a hypercube of dimensions $2, 3, \ldots, 17$ and measured the *load-factor*, which we define as the maximum load divided by the optimal load (i.e., M/N). Ideally, the load-factor is 1.

The result of the first experiment is shown in Figure 3. For a configuration in which each hypercube node hosts about 1000 guest nodes, the three algorithms *random walk*, *flip-bit*, and *naïve placement* do not differ significantly. For a smaller M/N, the naive placement method is slightly worse than the random walk and the flip-bit algorithms as expected. For the embedding of the tree in a hypercube with comparable size, the load-factor increases to 6.5 with naïve placement, 5.5 using the random walk, and 5 using the flip-bit algorithm.

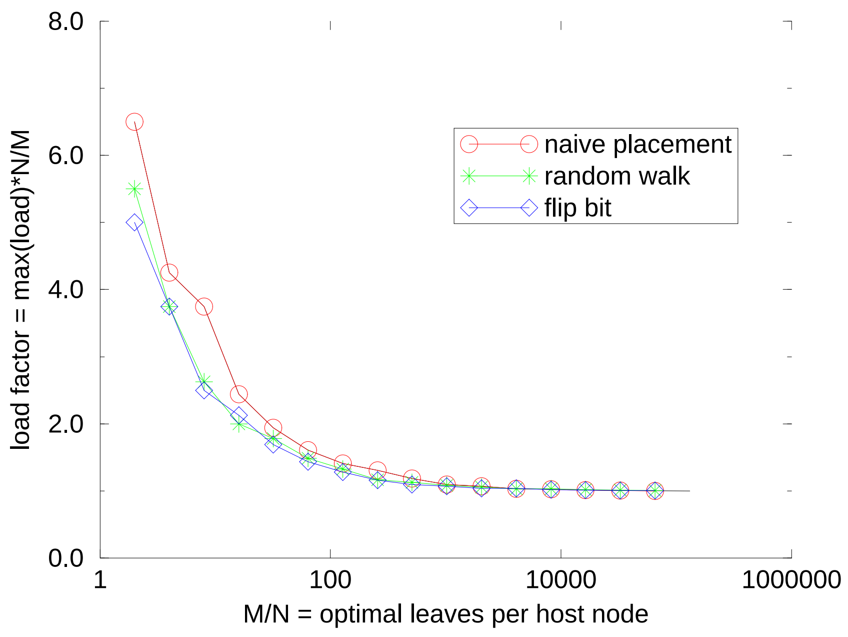

Figure 3: Maximum load for randomly generated binary trees

In the second experiment we embedded a complete binary with $2^{19} - 1$ nodes into the same hypercubes. By construction, the flip-bit algorithm distributes the nodes always perfectly, that is, every host node receives an equal share of guest nodes. Note that for both the random walk and the naïve placement algorithms, the number of bits inverted for each pair of nodes of the same height is equal. Therefore, all leaves are mapped on one half of the host hypercube, and the other half is only used by internal nodes. Therefore, the load-factor of both algorithms starts with 2 instead of 1, as evidenced in Figure 4. Let us note here that this anomaly is easy to fix, for example by allowing, as in the flip-bit algorithm, for children of a node to reside at the same host node as the parent (i.e., not inverting any bit). In this simulation the naïve placement performed as well as the random walk when the expected load per host node was around 500.

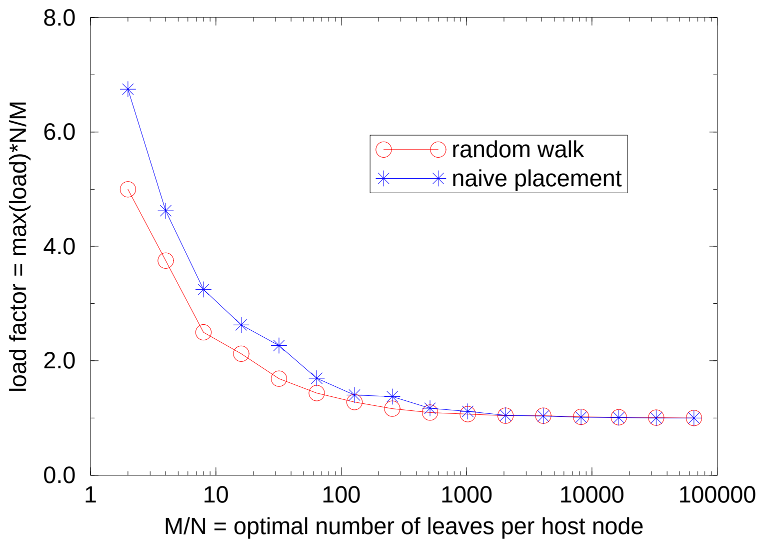

Figure 4: Maximum load for embedded complete binary trees

6 Conclusion

We have presented an analysis of the naïve placement algorithm for embedding large k-ary trees into fixed size hypercubes. The Matrix-tree theorem is used to show that equal load distribution of leaf nodes to host nodes is the only stationary distribution. The proof generalizes easily from hypercubes to arbitrary connected regular graphs.

The naïve placement algorithm was compared with two other embedding algorithms designed especially for small (complete) trees. Naïve placement performs as well as these other algorithms for larger tree sizes, and has the advantage of conceptual simplicity and ease of implementation.

In future work, we can consider generalizations in two directions: it would be of interest to look into different guest graphs than k-ary trees, for example irregular grids or similar graphs the occur in real life irregular computations. It would also be interesting to augment the asymptotic study done in this paper and find a quantification of the achieved load balancing as a function of the host/guest graph parameters.

Acknowledgment

We thank Sabine Öhring for much valued advice and fruitful discussions that helped creating this paper.

References

[BC89] S.N. Bhatt and Jin-Yi Cai. Take a walk, grow a tree. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 469–478, Los Alamitos, USA, 1989. IEEE Computer Society Press.

[BCLR92] S.N. Bhatt, F. R.K. Chung, F.T. Leighton, and A.L. Rosenberg. Efficient embeddings of trees in hypercubes. *SIAM Journal on Computing*, 21(1):151–162, 1992.

[Big93] N. Biggs. *Algebraic Graph Theory.* Cambridge University Press, 1993.

[HM96] V. Heun and E.W. Mayr. Efficient dynamic embeddings of arbitrary binary trees into hypercubes. In *Proceedings of the 3rd International Workshop on Parallel Algorithms for Irregularly Structured Problems (LNCS 1117)*, Santa Barbara, 1996.

[Lei92] F.T. Leighton. *Introduction to Parallel Algorithms and Architectures : Arrays – Trees – Hypercubes.* Morgan Kaufmann Publishers, San Mateo, CA, 1992.