Asymptotically Optimal Randomized Tree Embedding in Static Networks

Keqin Li
Department of Mathematics and Computer Science
State University of New York
New Paltz, New York 12561-2499
Email: li@mcs.newpaltz.edu

Abstract – *The problem of dynamic tree embedding in static networks is studied in this paper. We provide a unified framework for studying the performance of randomized tree embedding algorithms which allow a newly created tree node to take a random walk of a short distance to reach a processor nearby. In particular, we propose simple randomized algorithms on several most common and important static networks, including d-dimensional meshes, d-dimensional tori, and hypercubes. It is shown that these algorithms, which have a small constant dilation, are asymptotically optimal. Our analysis technique is based on random walks on static networks. Hence, analytical expressions for expected load on all the processors are available.*

Keywords: Asymptotic performance, dynamic load distribution, hypercube, mesh, randomized tree embedding, static network, torus.

1 Introduction

Many parallel computations are tree structured. Examples are divide-and-conquer algorithms, backtrack search algorithms, branch-and-bound computations, game-tree evaluation, and functional programs. Usually, the size and structure of a tree that represents a parallel computation is unpredictable at compiler-time. The tree grows gradually during the course of the computation. Initially, there is only one root process. As the computation proceeds, an existing process (or, a tree node) may create new processes, and these child processes should be immediately assigned to some processors for execution. In such a dynamic setting, the distribution of tree nodes over the processors is determined at runtime. It is desirable for a tree node to be assigned to a processor without knowing what the rest of the tree looks like.

Load distribution for a tree-structured parallel computation on a static network is also called dynamic tree embedding. There are two important performance measures of a load distribution, or, an embedding. The first one is the maximum load per processor, i.e., the maximum number of tree nodes that are assigned to a processor. The second measure is the dilation, i.e., the maximum distance between a pair of a parent node and a child node in the network. Both maximum load and dilation should be minimized such that the computation is evenly distributed among all the processors, and that communication overhead is kept at the minimum. It turns out that randomization is necessary if both the maximum load and dilation are to be minimized [8]. Randomized tree embedding algorithms and their performance analysis have been investigated by many researchers in recently years [1, 2, 3, 6, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 22, 23].

In this paper, we provide a unified framework for studying the performance of randomized tree embedding algorithms which allow a newly created tree node to take a random walk of a short distance to reach a processor nearby. In particular, we propose simple randomized algorithms on several most common and important static networks, including d-dimensional meshes, d-dimensional tori, and hypercubes. It is shown that these algorithms, which have a small constant dilation, are asymptotically optimal. Our analysis technique is based on random walks on static networks. Hence, analytical expressions for expected load on all the processors are available.

2 Definitions

Let $G = (V_G, E_G)$ be a directed graph that represents a static interconnection network, where $V_G = \{v_1, v_2, ..., v_N\}$ is a set of processors, and E_G is a set of unidirectional communication links. In many cases, a network is equivalent to an undirected graph. For these networks, each edge $\{v_i, v_j\}$ (i.e., a bidirectional communication link) is replaced by two arcs (v_i, v_j) and (v_j, v_i) (i.e., two unidirectional communication links). Normally, an interconnection network does not have self-loops, i.e., connections from a processor to itself. However, for convenience of discussion, we assume that each processor v_i has (a virtual) self-loop (v_i, v_i). If $(v_i, v_j) \in E_G$, we say that v_j is a neighbor of v_i. Every processor is a neighbor of itself. The number of neighbors of processor v_i, denoted by $\deg(v_i)$, is the outdegree of processor v_i.

A tree is divided into levels 0, 1, 2, ..., h, where h is the height of the tree. The root is on level 0, and a tree node is on level l if its distance from the root is l. We use M_l to represent the number of tree nodes on level l, where $0 \leq l \leq h$, and

$$M = M_0 + M_1 + M_2 + \cdots + M_h$$

the size of the tree.

Definition 1. A *randomized tree embedding algorithm* A can be represented as a matrix $P_A = (p_{ij})_{N \times N}$, such that

$p_{ij} \geq 0$ if $(v_i, v_j) \in E_G$, and $p_{ij} = 0$ if $(v_i, v_j) \notin E_G$, and that $p_{i1} + p_{i2} + \cdots + p_{iN} = 1$, where p_{ij} denotes the probability that processor v_i sends a new tree node to v_j. The matrix P_A is called the *load distribution matrix* of algorithm A. ∎

For example, if each processor sends a newly created process to one of the processor's neighbors, and these neighbors are chosen with equal probability, then we have $p_{ij} = 1/\deg(v_i)$, if $(v_i, v_j) \in E_G$. In general, it is possible that $p_{ii} > 0$, where $1 \leq i \leq N$.

3 General Discussion

Let $L_j^{(l)}$ denote the expected number of tree nodes on level l that are assigned to processor v_j with a randomly chosen initial processor. Let $\mathbf{r}^{(l)} = (r_1^{(l)}, r_2^{(l)}, ..., r_N^{(l)})$ be a vector, where

$$r_j^{(l)} = \frac{L_j^{(l)}}{M_l}, \quad 1 \leq j \leq N.$$

Clearly, we have

$$r_1^{(l)} + r_2^{(l)} + \cdots + r_N^{(l)} = 1.$$

If q_j is the probability that v_j is the initial processor, then $\mathbf{r}_0 = (q_1, q_2, ..., q_N)$.

Theorem 1. For any randomized tree embedding algorithm A whose load distribution matrix is P_A, and any initial processor selection policy $\mathbf{r}_0$, we have

$$\mathbf{r}^{(l)} = \mathbf{r}^{(0)} P_A^l,$$

for all $0 \leq l \leq h$, where P_A^l is the lth power of matrix P_A.

Proof. Let us consider the embedding of a tree T in a graph G. Assume that v_i is the initial processor. We examine the location v_j of a tree node c when a randomized embedding algorithm A is applied. Let c_0, c_1, c_2, ..., c_l be the path from the root of the tree to c, where c_0 is the root, and $c_l = c$. Also, assume that v_{k_0}, v_{k_1}, v_{k_2}, ..., v_{k_l} is a sequence of processors such that v_{k_u} is the processor that receives c_u, where $0 \leq u \leq l$, and $k_0 = i$, $k_l = j$. It is noticed that c is assigned to processor v_j in such a way that c "takes" a random walk from v_i of length l, and eventually reaches v_j:

$$v_{k_0} \to v_{k_1} \to v_{k_2} \to \cdots \to v_{k_l}.$$

Actually, the above random walk is not taken by c alone, it is taken by c_1, c_2, ..., c_l, one per step, as shown below:

$$v_{k_0} \stackrel{c_1}{\to} v_{k_1} \stackrel{c_2}{\to} v_{k_2} \to \cdots \stackrel{c_l}{\to} v_{k_l}.$$

The random walk described above can be specified by a Markov chain. The chain has N states s_1, s_2, ..., s_N, where s_j means that c is on processor v_j, $1 \leq j \leq N$. The matrix of transition probabilities of the Markov chain is exactly the same as P_A. Therefore, if $P_A^l = (p_{ij}^{(l)})_{N \times N}$, then $p_{ij}^{(l)}$ is the probability of a transition from state s_i to s_j in exactly l steps, that is, the probability that c (a tree node on level l) reaches v_j in a randomized embedding of the tree T on the network G, assuming that v_i is the initial processor.

Based on the above discussion, it is clear that processor v_j receives c with probability

$$\sum_{i=1}^{N} r_i^{(0)} p_{ij}^{(l)}, \quad \text{for all } 1 \leq j \leq N,$$

by considering all the possibilities of v_i. Since there are M_l nodes on level l, the expected number of tree nodes on level l that eventually reach v_j is

$$L_j^{(l)} = \left(\sum_{i=1}^{N} r_i^{(0)} p_{ij}^{(l)} \right) M_l,$$

that is,

$$r_j^{(l)} = \sum_{i=1}^{N} r_i^{(0)} p_{ij}^{(l)}, \quad \text{for all } 1 \leq j \leq N.$$

The last equation implies that $\mathbf{r}^{(l)} = \mathbf{r}^{(0)} P_A^l$. ∎

Let L_j denote the expected number of tree nodes that are assigned to processor v_j, that is,

$$L_j = L_j^{(0)} + L_j^{(1)} + L_j^{(2)} + \cdots + L_j^{(h)},$$

where $1 \leq j \leq N$. The following result is an immediate consequence of Theorem 1.

Corollary 1. L_j is given by

$$L_j = \sum_{l=0}^{h} L_j^{(l)} = \sum_{l=0}^{h} r_j^{(l)} M_l = \sum_{l=0}^{h} \left(\sum_{i=1}^{N} r_i^{(0)} p_{ij}^{(l)} \right) M_l,$$

for all $1 \leq j \leq N$. ∎

The performance measure of a randomized tree embedding generated by algorithm A is given below.

Definition 2. Consider the embedding of a tree with M nodes on a static network with N processors. Let $L_{MAX} = \max_{1 \leq j \leq N}(L_j)$ be the maximum expected load on all the processors. The *performance ratio* of algorithm A is $\alpha_A = L_{MAX}/(M/N)$, where M/N is the best load distribution achievable. Algorithm A is called *asymptotically optimal* if $\alpha_A \to 1$ as $h \to \infty$. ∎

4 Random Tree Models

There are a number of ways to model trees of arbitrary size and shape. For example, a complete-tree-based random tree is defined as follows [10, 11, 12]. Let the tree height h be bounded. The numbers of children of all the tree nodes on level l are independent and identically distributed random variables with mean $b_l \geq 0$, where $0 \leq l \leq h-1$. Tree nodes on level h do not have any child. It is clear that a complete-tree-based random tree is specified

by $(b_0, b_1, b_2, ..., b_{h-1})$. Using these quantities, we obtain $M_l = b_0 b_1 \cdots b_{l-1}$ for all $0 \le l \le h$, and

$$M = 1 + b_0 + b_0 b_1 + b_0 b_1 b_2 + \cdots + b_0 b_1 \cdots b_{h-1}.$$

In a reproduction tree [10, 11, 12], the numbers of children of all tree nodes are independent and identically distributed random variables with mean b, where $0 < b < 1$. It is clear that for a reproduction tree, $M_l = b^l$, for all $l \ge 0$, and

$$M = 1 + b + b^2 + b^3 + \cdots = \frac{1}{1-b}.$$

It turns out that our analysis has little to do with the actual specification of random trees. The key property of a tree that we need is given below.

Definition 3. A tree is said to be *healthy*, if as $h \to \infty$, we have

$$\frac{M_0 + M_1 + \cdots + M_l}{M} \to 0,$$

for all fixed $l \ge 0$. ∎

Essentially, a tree is healthy if its size is dominated by high levels as the tree height goes to infinity. If $b_l > 1$ for all $l \ge 0$, a complete-tree based random tree is healthy, though such a condition is not necessary. A reproduction tree is healthy for all $0 < b < 1$. Hence, the condition for a tree to be healthy is quite reasonable.

Theorem 2. Let A be a randomized tree embedding algorithm with load distribution matrix $P_A = (p_{ij})_{N \times N}$. If $\lim_{l \to \infty} p_{ij}^{(l)} = 1/N$, for all $1 \le i, j \le N$, then algorithm A is asymptotically optimal in embedding healthy trees. Such an optimality is independent of the initial processor selection.

Proof. Let v_i be an arbitrarily chosen initial processor. By Corollary 1,

$$L_j = \sum_{l=0}^{h} p_{ij}^{(l)} M_l,$$

for all $1 \le j \le N$. It is clear that for any small $\epsilon > 0$, there exists sufficiently large l_0 such that $p_{ij}^{(l)} \le 1/N + \epsilon$ for all $l > l_0$. Hence,

$$\begin{aligned} L_j &= \sum_{l=0}^{h} p_{ij}^{(l)} M_l \\ &= \sum_{l=0}^{l_0} p_{ij}^{(l)} M_l + \sum_{l=l_0+1}^{h} p_{ij}^{(l)} M_l \\ &\le \sum_{l=0}^{l_0} M_l + \left(\frac{1}{N} + \epsilon\right) \sum_{l=l_0+1}^{h} M_l, \end{aligned}$$

which implies that

$$\frac{\frac{L_j}{M}}{\frac{}{N}} \le \frac{\sum_{l=0}^{l_0} M_l + \left(\frac{1}{N} + \epsilon\right) \sum_{l=l_0+1}^{h} M_l}{\frac{M}{N}}.$$

Since the tree is healthy, the right side of the above inequality approaches $1 + N\epsilon$, as $h \to \infty$. Thus, $L_j/(M/N) \le 1 + N\epsilon$, for all $1 \le j \le N$, that is, $L_{MAX}/(M/N) \le 1 + N\epsilon$, or, $\alpha_A \le 1 + N\epsilon$, as $h \to \infty$. Since N is fixed, and ϵ can be arbitrarily close to zero, we conclude that $\alpha_A = 1$ as $h \to \infty$, i.e., algorithm A is asymptotically optimal. ∎

In the next few sections, we analyze a number of simple randomized tree embedding algorithms on meshes, tori, and hypercubes. We show that all these algorithms satisfy the condition in Theorem 2, and hence, are asymptotically optimal for healthy trees.

5 Linear Arrays

A linear array is a one-dimensional mesh $M_1 = (V_{M_1}, E_{M_1})$, where $V_{M_1} = \{v_1, v_2, ..., v_N\}$, and $(v_i, v_j) \in E_{M_1}$ if and only if $|i - j| = 1$, for $1 \le i, j \le N$.

Randomized tree growing on linear arrays can be studied in terms of *random walk with reflecting barriers* [4]. In such a random walk, there are N states s_1, s_2, ..., s_N, and the matrix of transition probabilities $P = (p_{ij})_{N \times N}$ is

$$P = \begin{bmatrix} q & p & 0 & 0 & \cdots & 0 & 0 & 0 \\ q & 0 & p & 0 & \cdots & 0 & 0 & 0 \\ 0 & q & 0 & p & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & q & 0 & p \\ 0 & 0 & 0 & 0 & \cdots & 0 & q & p \end{bmatrix}.$$

It is well known [4] that using the notations

$$x_i^{(k)} = \left(\frac{q}{p}\right)^{i/2} \sin\frac{\pi k i}{N} - \left(\frac{q}{p}\right)^{(i+1)/2} \sin\frac{\pi k (i-1)}{N},$$

and

$$y_j^{(k)} = \left(\frac{p}{q}\right)^{j/2} \sin\frac{\pi k j}{N} - \left(\frac{p}{q}\right)^{(j-1)/2} \sin\frac{\pi k (j-1)}{N},$$

for all $1 \le k \le N-1$, we have

$$\begin{aligned} p_{ij}^{(l)} = &\left[\sum_{k=0}^{N-1} \left(\frac{p}{q}\right)^k\right]^{-1} \left(\frac{p}{q}\right)^{j-1} \\ &+ \frac{2p}{N} \sum_{k=1}^{N-1} \frac{x_i^{(k)} y_j^{(k)} (2\sqrt{pq}\cos\pi k/N)^l}{1 - 2\sqrt{pq}\cos\pi k/N}, \end{aligned}$$

for all $1 \le i, j \le N$.

Now, let us consider a simple algorithm A_1 whose load distribution matrix is

$$P_{A_1} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & \cdots & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & \cdots & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & \cdots & 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix}.$$

In other words, in applying algorithm A_1, for $2 \leq i \leq N-1$, processor v_i sends a new tree node to one of its two neighbors with equal probability, and for $i = 1, N$, processor v_i sends a new tree node to its neighbor or holds the node with equal probability.

Theorem 3. For algorithm A_1, we have

$$p_{ij}^{(l)} = \frac{1}{N}\left(1 + \sum_{k=1}^{N-1} \frac{x_i^{(k)} y_j^{(k)} (\cos \pi k/N)^l}{1 - \cos \pi k/N}\right),$$

for all $1 \leq i, j \leq N$, where

$$x_i^{(k)} = \sin \frac{\pi k i}{N} - \sin \frac{\pi k (i-1)}{N},$$
$$y_j^{(k)} = \sin \frac{\pi k j}{N} - \sin \frac{\pi k (j-1)}{N},$$

for all $1 \leq k \leq N-1$. Furthermore,

$$\lim_{l \to \infty} p_{ij}^{(l)} = \frac{1}{N},$$

for all $1 \leq i, j \leq N$.

Proof. The first claim is a direct consequence of the above mentioned result of random walk. To see the second claim, we notice that $|\cos \pi k/N| < 1$, for all $1 \leq k \leq N-1$, and that the $x_i^{(k)}$'s and the $y_j^{(k)}$'s are independent of l. Hence, the summation on the right hand side of $p_{ij}^{(l)}$ approaches zero as l goes to infinity. ∎

6 Meshes

In a (two-dimensional) mesh network $M_2 = (V_{M_2}, E_{M_2})$ with $N = N_1 \times N_2$ processor:

$$V_{M_2} = \{v_{(i_1,i_2)} \mid 1 \leq i_1 \leq N_1,\ 1 \leq i_2 \leq N_2\}.$$

Two processors $v_{(i_1,i_2)}$ and $v_{(j_1,j_2)}$ are connected, i.e., $(v_{(i_1,i_2)}, v_{(j_1,j_2)}) \in E_{M_2}$, if and only if

$$|i_1 - j_1| + |i_2 - j_2| = 1,$$

for $1 \leq i_1, j_1 \leq N_1$, and $1 \leq i_2, j_2 \leq N_2$.

There are three types of processors in a mesh, namely, internal processors (they have four neighbors), boundary processors (they have three neighbors), and corner processors (they have two neighbors). Algorithm A_2 for randomized tree embedding on a mesh can be described as follows.

- An internal processor sends a new tree node to one of its four neighbors with equal probability $\frac{1}{4}$;
- A boundary processor sends a new tree node to one of its three neighbors with equal probability $\frac{1}{4}$, and holds the tree node with probability $\frac{1}{4}$;
- A corner processor sends a new tree node to one of its two neighbors with equal probability $\frac{1}{4}$, and holds the tree node with probability $\frac{1}{2}$.

Let $p_{(i_1,i_2)(j_1,j_2)}^{(l)}$ denote the probability that a tree node reaches processor $v_{(j_1,j_2)}$ from processor $v_{(i_1,i_2)}$ in a random walk of length l.

Theorem 4. For algorithm A_2, we have

$$p_{(i_1,i_2)(j_1,j_2)}^{(l)} = \frac{1}{2^l} \sum_{l_1+l_2=l} \binom{l}{l_1} p_{i_1 j_1}^{(l_1)} p_{i_2 j_2}^{(l_2)},$$

for all $1 \leq i_1, j_1 \leq N_1$, and $1 \leq i_2, j_2 \leq N_2$, where $p_{i_1 j_1}^{(l_1)}$ and $p_{i_2 j_2}^{(l_2)}$ are given in Theorem 3. Furthermore,

$$\lim_{l \to \infty} p_{(i_1,i_2)(j_1,j_2)}^{(l)} = \frac{1}{N},$$

for all $1 \leq i_1, j_1 \leq N_1$, and $1 \leq i_2, j_2 \leq N_2$.

Proof. For convenience, we say that a mesh with $N = N_1 \times N_2$ processor has N_2 rows and N_1 columns. A move from $v_{(i_1,i_2)}$ to $v_{(i_1 \pm 1, i_2)}$ is called a horizontal move, and a move from $v_{(i_1,i_2)}$ to $v_{(i_1,i_2 \pm 1)}$ is called a vertical move.

Also, for ease of analysis, we have the following special treatment to corner and boundary processors. Each corner processor has two self-loops, called the horizontal and the vertical self-loop respectively, and both transitions have probability $\frac{1}{4}$. A step along the horizontal (vertical, respectively) self-loop is considered as a horizontal (vertical, respectively) step. The reason to have separate self-loops is to allow a corner processor having two options for both horizontal and vertical moves, just like random walks in linear arrays. However, such a viewpoint does not change the actual behavior of a corner processor. The net effect is that a corner processor holds a new tree node (by taking either the horizontal or the vertical self-loop) with probability $\frac{1}{2}$.

For boundary processors, we assume that self-loops on processors $v_{(1,i_2)}$ and $v_{(N_1,i_2)}$, where $2 \leq i_2 \leq N_2 - 1$, are considered as horizontal loops, and self-loops on processors $v_{(i_1,1)}$ and $v_{(i_1,N_2)}$, where $2 \leq i_1 \leq N_1 - 1$, are considered as vertical loops.

Now, consider an l-step random walk taken by a tree node c. Assume that c takes l_1 horizontal steps and l_2 vertical steps, where $l_1 + l_2 = l$. Since during each step, c can take either a horizontal or a vertical step with probability $\frac{1}{2}$, such an event occurs with probability $\frac{1}{2^l}\binom{l}{l_1}$. Though these l_1 horizontal steps and l_2 vertical steps can be arbitrarily mixed, the net effect is that *c takes two independent random walks on two linear arrays M_1 and M_1' with N_1 and N_2 processors respectively, and such random walks are*

governed by algorithm A_1. For c to reach processor $v_{(j_1,j_2)}$ from $v_{(i_1,i_2)}$, under the condition that c takes l_1 horizontal steps and l_2 vertical steps, it is equivalent for c to reach v_{j_1} from v_{i_1} in the random walk on M_1, and to reach v_{j_2} from v_{i_2} in the random walk on M_1'. The above two events occurs independently with probabilities $p_{i_1j_1}^{(l_1)}$ and $p_{i_2j_2}^{(l_2)}$ respectively. Thus, the first claim follows the above argument.

The second claim is easily seen from Theorem 3. As $l \to \infty$, the probability that $l_1 \to \infty$ and $l_2 \to \infty$ is one. Since $\lim_{l_1\to\infty} p_{i_1j_1}^{(l_1)} = 1/N_1$, and $\lim_{l_2\to\infty} p_{i_2j_2}^{(l_2)} = 1/N_2$, we have

$$\lim_{l\to\infty} p_{(i_1,i_2)(j_1,j_2)}^{(l)} = \frac{1}{N_1N_2} \cdot \frac{1}{2^l} \sum_{l_1=0}^{l} \binom{l}{l_1} = \frac{1}{N},$$

for all $1 \le i_1, j_1 \le N_1$, and $1 \le i_2, j_2 \le N_2$. ∎

7 d-dimensional Meshes

In a d-dimensional mesh network $M_d = (V_{M_d}, E_{M_d})$ with $N = N_1 \times N_2 \times \cdots \times N_d$ processor:

$$V_{M_d} = \{v_{(i_1,i_2,...,i_d)} \mid 1 \le i_k \le N_k,\ 1 \le k \le d\}.$$

Two processors $v_{(i_1,i_2,...,i_d)}$ and $v_{(j_1,j_2,...,j_d)}$ are connected, i.e., $(v_{(i_1,i_2,...,i_d)}, v_{(j_1,j_2,...,j_d)}) \in E_{M_d}$, if and only if

$$\sum_{k=1}^{d} |i_k - j_k| = 1,$$

for $1 \le i_k, j_k \le N_k$, and $1 \le k \le d$.

The number of neighbors of a processor in M_d can be d, $d+1$, ..., $2d$. The randomized tree embedding algorithm A_d for M_d can be easily described as follows: processor $v_{(i_1,i_2,...,i_d)}$ sends a new tree node to one of its neighbors with probability $1/(2d)$, and keeps the node with probability $1 - \deg(v_{(i_1,i_2,...,i_d)})/2d$, where $\deg(v_{(i_1,i_2,...,i_d)})$ is the number of neighbors of processor $v_{(i_1,i_2,...,i_d)}$. (Notice that when $d = 1, 2$, algorithm A_d is exactly A_1 and A_2.)

Theorems 3 and 4 can be easily generalized as follows.

Theorem 5. For algorithm A_d, we have

$$p_{(i_1,i_2,...,i_d)(j_1,j_2,...,j_d)}^{(l)} = \frac{1}{d^l} \sum_{l_1+l_2+\cdots+l_d=l} \binom{l}{l_1, l_2, ..., l_d} p_{i_1j_1}^{(l_1)} p_{i_2j_2}^{(l_2)} \cdots p_{i_dj_d}^{(l_d)}$$

for all $1 \le i_k, j_k \le N_k$, and $1 \le k \le d$, where $p_{i_kj_k}^{(l_k)}$ is given in Theorem 3. Furthermore,

$$\lim_{l\to\infty} p_{(i_1,i_2,...,i_d)(j_1,j_2,...,j_d)}^{(l)} = \frac{1}{N},$$

for all $1 \le i_k, j_k \le N_k$, and $1 \le k \le d$.

Proof. The theorem is a straightforward extension of Theorem 4. ∎

8 d-dimensional Tori

Our approach to analyzing randomized tree embedding algorithms in meshes can also be applied to tori. In this section, we sketch our results for d-dimensional tori.

A ring is a one-dimensional torus $T_1 = (V_{T_1}, E_{T_1})$, where $V_{T_1} = \{v_0, v_1, ..., v_{N-1}\}$, and $(v_i, v_j) \in E_{T_1}$ if and only if $j = (i \pm 1) \bmod N$, for $1 \le i, j \le N$.

Randomized tree embedding on a ring can be studied in terms of a *cyclical random walk* with N states s_0, s_1, s_2, ..., s_{N-1}, whose matrix of transition probabilities is

$$Q = \begin{bmatrix} q_0 & q_1 & q_2 & \cdots & q_{N-2} & q_{N-1} \\ q_{N-1} & q_0 & q_1 & \cdots & q_{N-3} & q_{N-2} \\ q_{N-2} & q_{N-1} & q_0 & \cdots & q_{N-4} & q_{N-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ q_1 & q_2 & q_3 & \cdots & q_{N-1} & q_0 \end{bmatrix}.$$

From probability theory [4], we know that if $\theta = e^{2i\pi/N}$, which is the principle Nth root of unity, then we have

$$p_{ij}^{(l)} = \frac{1}{N} \sum_{k=0}^{N-1} \theta^{k(i-j)} t_k^l, \quad 1 \le i, j \le N,$$

where

$$t_k = \sum_{m=0}^{N-1} q_m \theta^{mk}, \quad 0 \le k \le N-1.$$

Now we analyze an algorithm B_1 in which, processor v_i sends a new tree node to one of its two neighbors, or keeps the tree node, with equal probability $\frac{1}{3}$, for all $0 \le i \le N-1$. Therefore, the load distribution matrix of algorithm B_1 is the same as Q, where $q_0 = q_1 = q_{N-1} = \frac{1}{3}$, and $q_m = 0$ for $m \ne 0, 1, N-1$.

Theorem 6. For algorithm B_1, we have

$$\begin{aligned} & p_{ij}^{(l)} \\ &= \frac{1}{N} \sum_{k=0}^{N-1} \theta^{k(i-j)} \left(\frac{1+\theta^k+\theta^{(N-1)k}}{3}\right)^l \\ &= \frac{1}{N} \sum_{k=0}^{N-1} \cos\frac{2k(i-j)\pi}{N} \left(\frac{1}{3}\left(1+2\cos\frac{2k\pi}{N}\right)\right)^l, \end{aligned}$$

for all $1 \le i, j \le N$. Furthermore,

$$\lim_{l\to\infty} p_{ij}^{(l)} = \frac{1}{N},$$

for all $1 \le i, j \le N$. ∎

The above result can easily be extended to d-dimensional tori $T_d = (V_{T_d}, E_{T_d})$ with $N = N_1 \times N_2 \times \cdots \times N_d$ processor:

$$V_{T_d} = \{v_{(i_1,i_2,...,i_d)} \mid 1 \le i_k \le N_k,\ 1 \le k \le d\},$$

where two processors $v_{(i_1,i_2,...,i_d)}$ and $v_{(j_1,j_2,...,j_d)}$ are connected, i.e., $(v_{(i_1,i_2,...,i_d)}, v_{(j_1,j_2,...,j_d)}) \in E_{T_d}$, if and only

if $j_k = (i_k \pm 1) \bmod N$, for exactly one k, $1 \le k \le d$, and $j_k = i_k$ for all other k. Algorithm B_1 is generalized to B_d for T_d, such that each processor sends a new tree node to one of its $2d$ neighbors with equal probability $1/(3d)$, or keeps the tree node with equal probability $\frac{1}{3}$.

Theorem 7. For algorithm B_d, we have

$$p^{(l)}_{(i_1,i_2,...,i_d)(j_1,j_2,...,j_d)} = \frac{1}{d^l} \sum_{l_1+l_2+\cdots+l_d=l} \binom{l}{l_1,l_2,...,l_d} p^{(l_1)}_{i_1 j_1} p^{(l_2)}_{i_2 j_2} \cdots p^{(l_d)}_{i_d j_d}$$

for all $1 \le i_k, j_k \le N_k$, and $1 \le k \le d$, where $p^{(l_k)}_{i_k j_k}$ is given in Theorem 6. Furthermore,

$$\lim_{l\to\infty} p^{(l)}_{(i_1,i_2,...,i_d)(j_1,j_2,...,j_d)} = \frac{1}{N},$$

for all $1 \le i_k, j_k \le N_k$, and $1 \le k \le d$. ∎

9 Hypercubes

It is noticed that when $N_1 = N_2 = \cdots = N_d = k$, a d-dimensional torus is also called k-ary d-cube. When $k = 2$, we get a binary d-cube, i.e., a d-dimensional hypercube. Clearly, algorithm B_d and Theorem 7 are applicable to a hypercube, where the $p^{(l_k)}_{i_k j_k}$'s are based on the powers of the matrix of transition probabilities

$$Q = \begin{bmatrix} \frac{1}{3} & \frac{2}{3} \\ \frac{2}{3} & \frac{1}{3} \end{bmatrix}$$

for a ring with $N_k = 2$ processors, for all $1 \le k \le d$. It is easy to verify that

$$Q^l = \begin{bmatrix} \frac{1}{2} + \frac{1}{2}(-\frac{1}{3})^l & \frac{1}{2} - \frac{1}{2}(-\frac{1}{3})^l \\ \frac{1}{2} - \frac{1}{2}(-\frac{1}{3})^l & \frac{1}{2} + \frac{1}{2}(-\frac{1}{3})^l \end{bmatrix},$$

for all $l \ge 0$.

In the following, we analyze algorithm C_d, which is slightly different algorithm B_d for a d-dimensional hypercube. To apply algorithm C_d, each processor $v_{(i_1,i_2,...,i_d)}$ sends a new tree node to one of its neighbors, i.e., $v_{(i_1,i_2,...\bar{i}_k,...,i_d)}$, with probability $1/(2d)$, where $1 \le k \le d$, or holds the tree node with probability $\frac{1}{2}$. Notice that the probability $\frac{1}{2}$ is split into d separate transitions, one with probability $1/(2d)$ for each of the d dimensions. Hence, it has the same probability $1/d$ to walk along all the dimensions in each step. In terms of Theorem 7, the $p^{(l_k)}_{i_k j_k}$'s are based on the matrix of transition probabilities

$$R = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

for a ring with $N_k = 2$ processors, for all $1 \le k \le d$. It is easy to see that

$$R^l = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix},$$

for all $l \ge 1$. Hence, by Theorem 7, algorithm C_d is asymptotically optimal, which comes from individual optimality in all the dimensions. However, we will prove this optimality formally.

Since a hypercube is a symmetric network, it makes sense to consider $p^{(l)}_{\delta}$, which is defined as $p^{(l)}_{(i_1,i_2,...,i_d)(j_1,j_2,...,j_d)}$ for pairs of processors $v_{(i_1,i_2,...,i_d)}$ and $v_{(j_1,j_2,...,j_d)}$ where the distance between the two processors is δ. Our goal is to obtain $p^{(l)}_{\delta}$, for all $0 \le \delta \le d$. Without loss of generality, let us assume that

$$(j_1, j_2, ..., j_d) = (\bar{i}_1, \bar{i}_2, ..., \bar{i}_\delta, i_{\delta+1}, i_{\delta+2}, ..., i_d),$$

that is, the indices differ in the first δ dimensions.

The following notations are introduced to obtain our analytical expressions for the $p^{(l)}_{\delta}$'s.

$$F_1(d,l) = \frac{1}{d^l} \sum_{\substack{l_1+l_2+\cdots+l_d=l \\ l_1,l_2,...,l_d>0}} \binom{l}{l_1,l_2,...,l_d} p^{(l_1)}_{i_1 j_1} p^{(l_2)}_{i_2 j_2} \cdots p^{(l_d)}_{i_d j_d}$$

is the probability that a tree node reaches $v_{(j_1,j_2,...,j_d)}$ from $v_{(i_1,i_2,...,i_d)}$ by taking a random walk of length l, such that $l_k > 0$ for all $1 \le k \le d$, where l_k is the number of steps along the kth dimension.

$$F_2(d,l) = p^{(l)}_{(i_1,i_2,...,i_d)(i_1,i_2,...,i_d)}$$

is the probability for a tree node to take a cycle from $v_{(i_1,i_2,...,i_d)}$ to $v_{(i_1,i_2,...,i_d)}$ in l steps. By the special values in R^l and the symmetry of a hypercube, the above quantities are independent of $(i_1, i_2, ..., i_d)$ and $(j_1, j_2, ..., j_d)$, i.e., the positions of the processors, but the length l of a walk, as explained below.

Now, let us examine the ways to calculate $F_1(d,l)$ and $F_2(d,l)$. In the definition of $F_1(d,l)$, we have $l_1, l_2, ..., l_d > 0$, which implies that $p^{(l_k)}_{i_k j_k} = \frac{1}{2}$, for all $1 \le k \le d$. Hence, we get

$$
\begin{aligned}
&F_1(d,l)\\
&= \frac{1}{d^l} \sum_{\substack{l_1+l_2+\cdots+l_d=l\\ l_1,l_2,\ldots,l_d>0}} \binom{l}{l_1,l_2,...,l_d} p_{i_1 j_1}^{(l_1)} p_{i_2 j_2}^{(l_2)} \cdots p_{i_d j_d}^{(l_d)}\\
&= \frac{1}{d^l 2^d} \sum_{\substack{l_1+l_2+\cdots+l_d=l\\ l_1,l_2,\ldots,l_d>0}} \binom{l}{l_1,l_2,...,l_d} \qquad (1)\\
&= \frac{1}{d^l 2^d} \sum_{z=0}^{d} (-1)^z \binom{d}{z} (d-z)^l \qquad (2)\\
&= \frac{1}{2^d} \sum_{z=0}^{d} (-1)^z \binom{d}{z} \left(1-\frac{z}{d}\right)^l.
\end{aligned}
$$

In simplifying the above summation (1), we notice that if there are z zeros in $l_1, l_2, ..., l_d$, with $\binom{d}{z}$ such combinations, and the remaining $(d-z)$ l_k's can be either zero or nonzero, then the summation is reduced to $(d-z)^l$. The equality (2) then follows the inclusion-exclusion principle.

As for $F_2(d,l)$, let us assume that $l_1 = l_2 = \cdots = l_z = 0$, and l_{z+1}, l_{z+2}, ..., $l_d > 0$. Then, we have $p_{i_k i_k}^{(l_k)} = 1$, for all $1 \le k \le z$, and $p_{i_k i_k}^{(l_k)} = \frac{1}{2}$, for all $z+1 \le k \le d$. Under the above condition, $F_2(d,l)$ is exactly $F_1(d-z,l)$. Since there are $\binom{d}{z}$ ways to choose z l_k's to be zero, and the probability for all these z l_k's to be zero is $(1-z/d)^l$, we obtain

$$
F_2(d,l) = \sum_{z=0}^{d} \binom{d}{z} \left(1-\frac{z}{d}\right)^l F_1(d-z,l).
$$

Notice that l_{z+1}, l_{z+2}, ..., $l_d > 0$ is guaranteed in $F_1(d-z,l)$,

(Remark: Our expressions for $F_1(d,l)$ and $F_2(d,l)$ are valid for $d > 0$. When $d = 0$, we should have $F_1(0,l) = F_2(0,l) = 1$ if $l = 0$, and $F_1(0,l) = F_2(0,l) = 0$ if $l > 0$.)

Theorem 8. For algorithm C_d, we have

$$
p_\delta^{(l)} = \sum_{l'=0}^{l} \binom{l}{l'} \left(\frac{\delta}{d}\right)^{l'} \left(1-\frac{\delta}{d}\right)^{l-l'} F_1(\delta,l') F_2(d-\delta, l-l'),
$$

for all $0 \le \delta \le d$, and $l \ge 0$. Furthermore,

$$
\lim_{l\to\infty} p_\delta^{(l)} = \frac{1}{N},
$$

for all $0 \le \delta \le d$.

Proof. We notice that

$$
\binom{l}{l'} \left(\frac{\delta}{d}\right)^{l'} \left(1-\frac{\delta}{d}\right)^{l-l'}
$$

is the probability of the event E that l' steps are taken along the first δ dimensions, and $l-l'$ steps are taken along the remaining $d-\delta$ dimensions. Under the condition that event E occurs, a tree node reaches $v_{(j_1,j_2,...,j_d)}$ from $v_{(i_1,i_2,...,i_d)}$ in l steps if the following two condition holds. First, at least one step must be taken along all the first δ dimensions; otherwise, the node cannot reach $v_{(j_1,j_2,...,j_d)}$, since $j_k = \bar{i}_k$, for all $1 \le k \le \delta$. Thus, this condition is satisfied with probability $F_1(\delta, l')$. Second, for the remaining $d-\delta$ dimensions, we need to reach $v_{(j_{\delta+1},j_{\delta+2},...,j_d)}$ from $v_{(i_{\delta+1},i_{\delta+2},...,i_d)}$. Since $j_k = i_k$, for all $\delta+1 \le k \le d$, this condition holds with probability $F_2(d-\delta, l-l')$.

For the second claim, it is easy to see that

$$
\lim_{l\to\infty} F_1(d,l) = \frac{1}{2^d},
$$

which implies that

$$
\lim_{l\to\infty} F_2(d,l) = \frac{1}{2^d},
$$

and

$$
\begin{aligned}
&\lim_{l\to\infty} p_\delta^{(l)}\\
&= \lim_{l\to\infty} \sum_{l'=0}^{l} \binom{l}{l'} \left(\frac{\delta}{d}\right)^{l'} \left(1-\frac{\delta}{d}\right)^{l-l'}\\
&\qquad\qquad F_1(\delta,l') F_2(d-\delta,l-l')\\
&= \lim_{l\to\infty} \sum_{l'=0}^{l} \binom{l}{l'} \left(\frac{\delta}{d}\right)^{l'} \left(1-\frac{\delta}{d}\right)^{l-l'} \frac{1}{2^\delta} \frac{1}{2^{d-\delta}}\\
&= \frac{1}{2^d} \left(\lim_{l\to\infty} \sum_{l'=0}^{l} \binom{l}{l'} \left(\frac{\delta}{d}\right)^{l'} \left(1-\frac{\delta}{d}\right)^{l-l'} \right)\\
&= \frac{1}{2^d},
\end{aligned}
$$

for all $0 \le \delta \le d$. ∎

10 Dilation-Δ Embedding

Our discussion so far is restricted to dilation-1 embeddings. It turns out that the extension to dilation-Δ embeddings is straightforward.

Definition 4. For any randomized tree embedding algorithm A given in Definition 1, algorithm A_Δ, where $\Delta \ge 1$, is a randomized tree embedding algorithm which allows a newly created tree node to take a random walk of length Δ, and each step is governed by algorithm A. ∎

As a matter of fact, such an algorithm A_Δ can also be specified by a load distribution matrix $P_{A_\Delta} = (q_{ij})_{N\times N}$, where q_{ij} stands for the probability that a tree node created on v_i reaches v_j after algorithm A_Δ is applied.

Theorem 9. For all algorithm A, and $\Delta \ge 1$, we have $P_{A_\Delta} = P_A^\Delta$.

Proof. Let the load distribution matrix of algorithm A be $P_A = (p_{ij})_{N\times N}$. It is clear that q_{ij}, a one step transition

probability in A_Δ, is actually the higher transition probability $p_{ij}^{(\Delta)}$ in A. Since $q_{ij} = p_{ij}^{(\Delta)}$, for all $1 \leq i, j \leq N$, we get $P_{A_\Delta} = P_A^\Delta$. ∎

Theorem 9 implies that for an asymptotically optimal algorithm A, A_Δ may spread tree nodes much faster than the original algorithm A. In other words, a larger dilation results in better tree node distribution.

11 Final Remarks

The problem of dynamic tree embedding in static networks is studied in this paper. We provide a unified framework for studying the performance of randomized tree embedding algorithms which allow a newly created tree node to take a random walk of a short distance to reach a processor nearby. In particular, we propose simple randomized algorithms on several most common and important static networks, including d-dimensional meshes, d-dimensional tori, and hypercubes. It is shown that these algorithms, which have a small constant dilation, are asymptotically optimal. Our analysis technique is based on random walks on static networks. Hence, analytical expressions for expected load on all the processors are available.

Acknowledgment

The author's research was partially supported by the NASA/University Joint Venture in Space Science Program of National Aeronautics and Space Administration and the Research Foundation of State University of New York under Grant NAG8-1313.

References

[1] W. Aiello and F.T. Leighton, "Coding theory, hypercube embeddings, and fault tolerance," *Proc. of ACM Symp. on Parallel Algorithms and Architectures*, 1991.

[2] S. Bhatt and J.-Y. Cai, "Taking random walks to grow trees in hypercubes," *Journal of the ACM*, vol.40, no.3, pp.741-764, 1993. (An earlier version appeared as "Take a walk, grow a tree," *Proc. of IEEE Symp. on Foundations of Computer Science*, pp.469-478, 1988.)

[3] S. Bhatt, D. Greenberg, T. Leighton, and P. Liu, "Tight bounds for on-line tree embeddings," *Proc. of 2nd ACM-SIAM Symp. on Discrete Algorithms*, pp.344-350, 1991.

[4] W. Feller, *An Introduction to Probability Theory and Its Applications*, Vol. I, 3rd ed., John Wiley & Sons, New York, 1968.

[5] T. Harris, *The Theory of Branching Processes*, Springer, Berlin, 1963.

[6] C. Kaklamanis and G. Persiano, "Branch-and-bound and backtrack search on mesh-connected arrays of processors," *Proc. of ACM Symp. on Parallel Algorithms and Architectures*, pp.118-126, 1992.

[7] R.M. Karp and Y. Zhang, "Randomized parallel algorithms for backtrack search and branch-and-bound computation," *Journal of the ACM*, vol.40, no.3, pp.765-789, 1993.

[8] F.T. Leighton, M.J. Newman, A.G. Ranade, and E.J. Schwabe, "Dynamic tree embeddings in butterflies and hypercubes," *SIAM Journal on Computing*, vol.21, no.4, pp.639-654, 1992.

[9] F.T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays · Trees · Hypercubes*, Morgan Kaufmann, 1992.

[10] K. Li, "Maintenance of tree structured computations on parallel and distributed computer systems," *Proc. of the 11th Annual ACM Symposium on Applied Computing*, pp.337-343, February 1996.

[11] K. Li, "On dynamic tree growing in hypercubes," *Proc. of the 12th Annual ACM Symposium on Applied Computing*, pp.496-503, February 1997.

[12] K. Li, "Determining the expected load of dynamic tree embeddings in hypercubes," *Proc. of the 17th International Conference on Distributed Computing Systems*, pp.508-515, May 1997.

[13] K. Li, "A randomized algorithm for dynamic tree growing on k-ary n-cubes," *Proc. of International Conference on Parallel and Distributed Processing Techniques and Applications*, vol. II, pp.600-609, June 1997.

[14] K. Li, "Analysis of randomized load distribution for reproduction trees in linear arrays and rings," *Proc. of 11th Annual International Symposium on High Performance Computing Systems*, pp.199-215, July 1997.

[15] K. Li, "Barrel shifter – a close approximation to the completely connected network in supporting dynamic tree structured computations," *Proc. of 49th IEEE National Aerospace and Electronics Conference*, pp.202-215, July 1997.

[16] K. Li, "Performance modeling and analysis of dynamic tree embedding in mesh networks," *Proc. of the 9th International Conference on Parallel and Distributed Computing and Systems*, pp.470-475, October 1997.

[17] K. Li, "Performance evaluation of probabilistic tree embedding in cube-connected cycles," *Proc. of the 13th Annual ACM Symposium on Applied Computing*, February 1998.

[18] K. Li, "Analyzing the performance of a probabilistic algorithm for embedding reproduction trees in static networks," *Proc. of the 6th High Performance Computing Symposium*, April 1998.

[19] K. Li, Y. Pan, H. Shen, G.H. Young, and S.-Q. Zheng, "Lower bounds for dynamic tree embedding in bipartite graphs," Technical Report #96-001, Department of Computer Science, Louisiana State University, Baton Rouge, LA, 1996.

[20] M.A. Palis and D.S.L. Wei, "Backtracking and branch-and-bound on mesh-connected computers with reconfigurable buses," *Proc. of 7th International Conference on Parallel and Distributed Computing and Systems*, pp. 243-247, 1995.

[21] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley, 1984.

[22] A.G. Ranade, "Optimal speedup for backtrack search on a butterfly network," *Proc. of the 3rd ACM Symp. on Parallel Algorithms and Architectures*, pp.40-48, 1991.

[23] H. Shen, K. Li, Y. Pan, G.H. Young, and S.-Q. Zheng, "Performance analysis for dynamic tree embedding in k-partite networks by random walk," *Proceedings of IEEE International Symposium on Parallel Architectures, Algorithms, and Networks*, December 1997.

[24] F. Spitzer, *Principles of Random Walk*, D. Van Nostrand Company, Inc., Princeton, New Jersey, 1964.