Performance Evaluation of a Random-Walk-Based Algorithm for Embedding Dynamically Evolving Trees in Butterflies

Keqin Li
Department of Computer Science
State University of New York
New Paltz, New York 12561-2440
Email: li@mcs.newpaltz.edu

Abstract

In many tree-structured parallel computations, the size and structure of a tree that represents a parallel computation is unpredictable at compile-time; the tree evolves gradually during the course of the computation. When such a computation is performed on a static network, the dynamic tree embedding problem is to distribute the tree nodes to the processors of the network such that all the processors receive roughly the same amount of load and that communicating nodes are assigned to neighboring processors. Furthermore, when a new tree node is generated, it should be immediately assigned to a processor for execution without any information on the further evolving of the tree; and load distribution is performed by all processors in a totally distributed fashion.

We study the problem of embedding dynamically evolving trees in butterflies. We evaluate the performance of a random-walk-based algorithm. Our performance data demonstrate that butterflies have comparable performance with hypercubes in supporting tree-structured parallel computations.

1 Introduction

1.1 Problem Definition

High performance parallel computing requires distributing processes in a parallel computation over processors in a parallel computer such that all the processors receive roughly the same amount of load and that communicating processes are assigned to neighboring processors. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a task interaction graph which represents a parallel computation, where $\mathcal{V} = \{v_1, v_2, ..., v_M\}$ is a set of M processes (i.e., tasks), and $\mathcal{E}$ is a set of edges that stand for intertask communications. Let $\mathcal{N} = (\mathcal{P}, \mathcal{C})$ be a graph which represents a static network, where $\mathcal{P} = \{P_0, P_1, P_2, ..., P_{N-1}\}$ is a set of N processors, and $\mathcal{C}$ is a set of interprocessor connections. The load distribution problem is to embed a task interaction graph into a static network. An *embedding* is actually a mapping function $\phi : \mathcal{V} \to \mathcal{P}$ from the set of processes to the set of processors, such that $\phi(v_i)$ is the processor on which process v_i is executed.

The embedding problem can be classified into two categories, namely, static embedding and dynamic embedding. *Static embedding* is done at compile-time, where the complete task interaction graph $\mathcal{G}$ is given. *Dynamic embedding* is performed at run-time, where the task interaction graph $\mathcal{G}$ is not known in advance, and it evolves and expands gradually during the course of a parallel computation. It is more difficult to achieve good performance in dynamic embedding than in static embedding. While static embedding of various task graphs into various networks has been extensively studied (see [33] and the references therein), a little work has been done for dynamic embedding.

Many parallel computations are tree-structured (i.e., where $\mathcal{G}$ is a tree). Examples are divide-and-conquer algorithms, backtrack searching, branch-and-bound computations, game-tree evaluation, functional and logical programming, and various numeric computations. In these tree-structured parallel computations, the size and structure of a tree that represents a parallel computation is unpredictable at compile-time. The tree evolves gradually during the course of the computation. Initially, there is one process, i.e., the root of a tree. This process could reside on any processor (called the initial or origin processor). As the computation proceeds, existing processes (i.e., tree nodes) create new processes, and these child processes should be assigned immediately to some processors for execution according to a mapping function ϕ.

The above applications motivate the problem of *dynamic tree embedding* (or, dynamic load distribution for tree-structured computations on static networks). In such a dynamic setting, the mapping function ϕ is not available in advance and must be calculated at run-time. It is desirable for a tree node to be mapped to a processor without any information on the further evolving of the tree. When a new tree node is generated, it should be immediately assigned to a processor for execution. When a tree node v_{i_1} executing on processor P_{j_1} spawns a new node v_{i_2}, P_{j_1} should select a processor $P_{j_2} = \phi(v_{i_2})$ and send v_{i_2} to P_{j_2} for processing. Such a decision is made locally on P_{j_1} without consulting

to other processors. Therefore, the load distribution function ϕ is computed by all processors in a *totally distributed* fashion. It is required that the mapping function ϕ is easy to calculate for fast load distribution and low system cost. It is also required that there is no centralized control mechanism, no centralized task dispatcher, no global load status record, no exchange of current load status information among the processors, and no process migration (i.e., once a tree node v_i is assigned to processor $\phi(v_i)$, v_i cannot move to another processor).

There are two important performance considerations in all embedding problems. The first one is the *maximum load* per processor. The load L_j on a processor P_j is the number of tree nodes assigned to P_j, i.e., $L_j = |\{v_i \mid \phi(v_i) = P_j\}|$. Clearly, an embedding ϕ should minimize the maximum load $L_{MAX} = \max_{0 \le j < N}(L_j)$, such that all the processors perform approximately equal amount of computation. In other words, the M tree nodes should be distributed over the N processors as evenly as possible, such that all the computing power are efficiently utilized. Another important consideration is the *dilation* Δ of an embedding, that is, the maximum distance between a pair of communicating processes in the network. Formally, we have $\Delta = \max_{(v_{i_1}, v_{i_2}) \in \mathcal{E}}(\mathrm{dist}(\phi(v_{i_1}), \phi(v_{i_2})))$, where $\mathrm{dist}(P_{j_1}, P_{j_2})$ is the distance between two processors in the network $\mathcal{N}$. The dilation affects communication overhead and should also be minimized. Small dilation provides high communication locality. Hence, processes that need to communicate to each other during the computation should be assigned to processors that are close to each other.

Based on the fact that there are a wide spectrum of tree-structured applications in computer science and engineering, and the wide availability of distributed memory multicomputers with static interconnection networks, dynamic tree embedding algorithms with high performance are definitely very important.

1.2 Randomized Tree Embedding

A number of leading scientists pioneered the research in this field. Bhatt and Cai [2] were the first to raise the problem of dynamic tree embedding in static networks (hypercubes in [2]). They proposed and analyzed a *randomized* tree embedding strategy for maintaining dynamically evolving binary trees on hypercube networks. The algorithm is random-walk-based and also uses local rearrangement. For arbitrary binary trees, their algorithm achieves optimal load $O(M/N)$ and dilation $O(\log \log N)$ with high probability [3].

Since then, dynamic tree growing algorithms have been investigated by several researchers in recent years. Randomization is justified by observing that minimization of the maximum load and dilation are conflicting requirements. In particular, Leighton *et al.* showed that any deterministic algorithm which dynamically embeds an M-node binary tree in an N-node hypercube, such that the maximum load is $O(M/N)$, must have not only maximum but also average dilation $\Omega(\sqrt{\log N})$ [13]. Therefore, a dynamic tree embedding algorithm that simultaneously minimizes maximum load and dilation (within constant factors) must be randomized.

For embedding in hypercubes and butterflies, Leighton *et al.* also reduced the dilation to $O(1)$ and achieved optimal load and dilation simultaneously within constant factors. Aiello and Leighton [1] considered the congestion of embeddings, i.e., the maximum number of tree edges that are routed through any single communication link of a hypercube. Bhatt *et al.* [4] studied the trade-offs between load imbalance and congestion.

Dynamically evolving trees have also been treated from other points of view. Karp and Zhang presented randomized parallel backtrack and branch-and-bound algorithms on completely connected networks that run in optimal time with high probability [12]. Ranade strengthened the result by considering butterflies which are bounded degree networks [36]. Instead of embedding trees with small dilation and even load distribution, the aim of their research is to minimize the parallel execution time. These pioneer studies have been followed by a number of other researchers [6, 8, 10, 11, 34].

1.3 Our Approach

The randomized tree embedding algorithms, which we have been studying and will continue to investigate, are random-walk-based. Essentially, such an algorithm allows a newly created process to take a random walk of short distance to reach a processor nearby. Formally, when a new tree node v_i is created on processor P_{j_0}, the node v_i is allowed to take a random walk of Δ steps: $P_{j_0} \to P_{j_1} \to P_{j_2} \to \cdots \to P_{j_\Delta}$. Let $\deg(\mathcal{N}, j)$ denote the degree of processor P_j in $\mathcal{N}$. Then, for all $1 \le k \le \Delta$, P_{j_k} is selected among all the $\deg(\mathcal{N}, j_{k-1})$ neighbors of $P_{j_{k-1}}$ with equal probability. When process v_i reaches P_{j_Δ}, it is assigned to P_{j_Δ} and executed on P_{j_Δ}, i.e., $\phi(v_i) = P_{j_\Delta}$. We call this algorithm the *Canonical Algorithm*. There are many variations of the Canonical Algorithm which are worth of further investigation (see, e.g., [5, 29]).

It is clear that by using the above randomized tree embedding algorithm, the distance between each pair of parent/child is at most Δ. Hence, the algorithm generates a dilation-Δ embedding. The value Δ can be chosen as a small constant. (We do not allow process migration, since this may increase dilation arbitrarily.) The remaining concern is to analyze the maximum load L_{MAX}. The performance of a randomized tree embedding can be measured as follows. Let M denote the tree size, and N the number of processors in a network $\mathcal{N}$. Assume that by using a tree embedding algorithm, the expected number of tree nodes assigned to processor P_j is L_j. We are interested in the maximum expected load $L_{MAX} = \max(L_0, L_1, ..., L_{N-1})$. Let L_{OPT} be the best possible maximum expected load on all the processors, i.e., $L_{OPT} = M/N$. The *performance ratio* of a randomized tree embedding is $\alpha = L_{MAX}/L_{OPT}$. If we consider the performance ratio α_h as a function of tree height h, then $\alpha_\infty = \lim_{h \to \infty} \alpha_h$ is called the *asymptotic performance ratio*.

The performance ratio α is determined by the number of processors N, the topology of the network $\mathcal{N}$, the length of a random walk Δ, the number of tree nodes M, and the model of random trees, and even the choice of the initial processor. The performance ratio α is expected to be as close to one as possible. It is shown in [16] that the expected execution time of a parallel computation supported by a randomized tree embedding algorithm is no more than $\alpha + 1$ times the optimal execution time.

We have discovered a method to analyze the performance of random-walk-based randomized tree embedding algorithms in static interconnection networks. Our method employs recurrence relations and linear systems of equations that characterize the expected load on each processor. By using these recurrence relations and linear systems of equations, we can easily obtain numerical data which are quite illustrative and convincing, make general observations from these data, and prove these observations analytically. The method was initially developed for hypercubes [16, 17]. However, it turns out that the analysis methodology can be extended, generalized, and applied to the analysis of dynamic tree embedding in arbitrary networks. Previous analysis mentioned above are all network-dependent. It is still likely that *ad hoc* network-dependent analysis techniques will be developed to handle different network topologies. However, our method is versatile, network-independent, and readily to be applied to various networks. In this sense, our novel methodology is substantially different from previous approaches and is worth of more attention and further investigation. The present paper makes new effort towards this direction. In particular, we apply the Canonical Algorithm to butterfly networks.

2 Tree Models

To evaluate the performance of randomized tree embedding, we need models of random trees. A number of random tree models have been proposed in [15, 16, 17]. References [7, 9, 15, 16, 17, 35] contain for more discussion on these tree models. Here, we only give the definitions of the following deterministic and probabilistic tree models.

- In a *Complete Tree* with branching factor b and height h, every node has b children. Nodes on level h are leaves.
- A *Complete-Tree-Based Random Tree* has a fixed hight h. The number of children of all tree nodes on level l, where $0 \le l \le h-1$, are independent and identically (i.i.d.) random variables with an arbitrary probability distribution $(u_{l,0}, u_{l,1}, u_{l,2}, ..., u_{l,b}, ...)$, where $u_{l,b}$, $b \ge 0$, is the probability that a node on level l spawns b children. A node on level h must be a leaf node, and it cannot create new processes.
- The height h of a *Randomized Complete Tree* is a random variable with an arbitrary probability distribution $(v_0, v_1, v_2, ..., v_h, ...)$, where v_h is the probability that the tree height is $h \ge 0$. Under the condition that the tree height is h, the number of children of all tree nodes on level l, where $0 \le l \le h-1$, are i.i.d. random variables with an arbitrary probability distribution $(u_{h,l,0}, u_{h,l,1}, u_{h,l,2}, ..., u_{h,l,b}, ...)$.
- In a *Reproduction Tree*, the number of children of all tree nodes are i.i.d. random variables with an arbitrary probability distribution $(u_0, u_1, u_2, ..., u_b, ...)$, with mean $\bar{b} = u_1 + 2u_2 + 3u_3 + \cdots$, and $\bar{b} < 1$.

Our research so far has been primarily focused on complete trees and reproduction trees. However, extensions to complete-tree-based random trees and randomized complete trees are straightforward.

3 The Algorithm and Analysis

Consider the class of b-ary complete trees. Let $\mathcal{N} = (\mathcal{P}, \mathcal{C})$ be an arbitrary network. For an asymmetric network $\mathcal{N}$, the quality of an embedding depends on the initial processor, i.e., where the root process resides. Let us define $L_{\mathcal{N}}(h, \delta, j_1, j_2)$, where $h \ge 0$, $0 \le \delta \le \Delta$, $0 \le j_1, j_2 < N$, to be the expected load on processor P_{j_2} in a dilation-Δ embedding (produced by the Canonical Algorithm) of a complete b-ary tree with height h in network $\mathcal{N}$, when P_{j_1} is the initial processor, and the root has δ more steps to go in its random walk. Let $\eta(j,1)$, $\eta(j,2)$, ..., $\eta(j, \deg(\mathcal{N}, j))$ denote the indices of the $\deg(\mathcal{N}, j)$ neighbors of processor P_j. The following theorem, originally proved in [21], provides a general method for analyzing the embedding of b-ary complete trees (produced by the Canonical Algorithm) in general networks.

Theorem 1. $L_{\mathcal{N}}(h, \delta, j_1, j_2)$ satisfies the following recurrence relation:

$$
\begin{aligned}
&L_{\mathcal{N}}(0,0,j_1,j_1) = 1, \quad 0 \le j_1 \le N-1;\\
&L_{\mathcal{N}}(0,0,j_1,j_2) = 0, \quad 0 \le j_1 \ne j_2 \le N-1;\\
&L_{\mathcal{N}}(h,0,j_1,j_1) = 1 + bL_{\mathcal{N}}(h-1,\Delta,j_1,j_1),\\
&\qquad\qquad h \ge 1,\ 0 \le j_1 \le N-1;\\
&L_{\mathcal{N}}(h,0,j_1,j_2) = bL_{\mathcal{N}}(h-1,\Delta,j_1,j_2),\\
&\qquad\qquad h \ge 1,\ 0 \le j_1 \ne j_2 \le N-1;\\
&L_{\mathcal{N}}(h,\delta,j_1,j_2)\\
&\quad = \frac{1}{\deg(\mathcal{N},j_1)} \sum_{k=1}^{\deg(\mathcal{N},j_1)} L_{\mathcal{N}}(h,\delta-1,\eta(j_1,k),j_2),\\
&\qquad\qquad h \ge 0,\ 1 \le \delta \le \Delta,\ 0 \le j_1, j_2 \le N-1.
\end{aligned}
$$

The recurrence relation given in Theorem 1 provides a systematic way to calculate the expected load on each processor in the style of dynamic programming [32]. For $h \ge 0$ and $0 \le \delta \le \Delta$, define $\mathcal{L}_{h,\delta}$ to be

$$
\begin{bmatrix}
L_{\mathcal{N}}(h,\delta,0,0) & L_{\mathcal{N}}(h,\delta,0,1) & \cdots & L_{\mathcal{N}}(h,\delta,0,N-1)\\
L_{\mathcal{N}}(h,\delta,1,0) & L_{\mathcal{N}}(h,\delta,1,1) & \cdots & L_{\mathcal{N}}(h,\delta,1,N-1)\\
\vdots & \vdots & \ddots & \vdots\\
L_{\mathcal{N}}(h,\delta,N-1,0) & L_{\mathcal{N}}(h,\delta,N-1,1) & \cdots & L_{\mathcal{N}}(h,\delta,N-1,N-1)
\end{bmatrix}.
$$

Then the computation proceeds as follows:

$$\begin{array}{ccccccccc}
\mathcal{L}_{0,0} & \rightarrow & \mathcal{L}_{0,1} & \rightarrow & \mathcal{L}_{0,2} & \rightarrow & \cdots & \rightarrow & \mathcal{L}_{0,\Delta} \\
\mathcal{L}_{1,0} & \rightarrow & \mathcal{L}_{1,1} & \rightarrow & \mathcal{L}_{1,2} & \rightarrow & \cdots & \rightarrow & \mathcal{L}_{1,\Delta} \\
\mathcal{L}_{2,0} & \rightarrow & \mathcal{L}_{2,1} & \rightarrow & \mathcal{L}_{2,2} & \rightarrow & \cdots & \rightarrow & \mathcal{L}_{2,\Delta} \\
\vdots & & \vdots & & \vdots & & \cdots & & \vdots \\
\mathcal{L}_{h-1,0} & \rightarrow & \mathcal{L}_{h-1,1} & \rightarrow & \mathcal{L}_{h-1,2} & \rightarrow & \cdots & \rightarrow & \mathcal{L}_{h-1,\Delta} \\
\mathcal{L}_{h,0} & & & & & & & &
\end{array}$$

where $\mathcal{L}_{0,0}$ is the identity matrix. This is a two dimensional recurrence relation, with our desired results given by $\mathcal{L}_{h,0}$, which provides the expected loads on all processors. Therefore, the computational complexity of this procedure is $O(N^2\Delta\deg(\mathcal{N})h) = O(N^2\Delta\deg(\mathcal{N})\log M)$ to calculate $\mathcal{L}_{h,0}$, where $\deg(\mathcal{N}) = \max_{0\le j\le N-1}(\deg(\mathcal{N}, j))$ is the maximum degree of all processors. Moreover, we only need to keep $\mathcal{L}_{h,\delta-1}$ when we calculate $\mathcal{L}_{h,\delta}$, that is, the number of variables is $O(N^2)$.

The reproduction tree model is handled in a different way. Let $\bar{L}_{\mathcal{N}}(\delta, j_1, j_2)$ where $0 \le \delta \le \Delta$, $0 \le j_1, j_2 < N$, be the expected load on processor P_{j_2} in a dilation-Δ embedding (produced by the Canonical Algorithm) of a reproduction tree in network $\mathcal{N}$, when P_{j_1} is the initial processor, and the root of the tree has δ more steps to go. The following theorem, originally proved in [21], provides a general method for analyzing the embedding of reproduction trees (produced by the Canonical Algorithm) in general networks.

Theorem 2. $\bar{L}_{\mathcal{N}}(\delta, j_1, j_2)$ satisfies the following linear system of equations:

$$\begin{aligned}
&\bar{L}_{\mathcal{N}}(0, j_1, j_1) = 1 + \bar{b}\bar{L}_{\mathcal{N}}(\Delta, j_1, j_1), \quad 0 \le j_1 \le N-1; \\
&\bar{L}_{\mathcal{N}}(0, j_1, j_2) = \bar{b}\bar{L}_{\mathcal{N}}(\Delta, j_1, j_2), \quad 0 \le j_1 \ne j_2 \le N-1; \\
&\bar{L}_{\mathcal{N}}(\delta, j_1, j_2) \\
&\quad = \frac{1}{\deg(\mathcal{N}, j_1)} \sum_{k=1}^{\deg(\mathcal{N}, j_1)} \bar{L}_{\mathcal{N}}(\delta-1, \eta(j_1, k), j_2), \\
&\qquad 1 \le \delta \le \Delta,\ 0 \le j_1, j_2 \le N-1.
\end{aligned}$$

The equations in Theorem 2 comprise a linear system of equations with $(\Delta+1)N^2$ variables. A closed form solution to the above linear system of equation seems difficult to find in general. However, numerical solutions can be found easily using an iterative technique. For $0 \le \delta \le \Delta$, define $\bar{\mathcal{L}}_\delta$ to be

$$\begin{bmatrix}
\bar{L}_{\mathcal{N}}(\delta,0,0) & \bar{L}_{\mathcal{N}}(\delta,0,1) & \cdots & \bar{L}_{\mathcal{N}}(\delta,0,N-1) \\
\bar{L}_{\mathcal{N}}(\delta,1,0) & \bar{L}_{\mathcal{N}}(\delta,1,1) & \cdots & \bar{L}_{\mathcal{N}}(\delta,1,N-1) \\
\vdots & \vdots & \ddots & \vdots \\
\bar{L}_{\mathcal{N}}(\delta,N-1,0) & \bar{L}_{\mathcal{N}}(\delta,N-1,1) & \cdots & \bar{L}_{\mathcal{N}}(\delta,N-1,N-1)
\end{bmatrix}.$$

The computation proceeds as follows. First, the matrix $\bar{\mathcal{L}}_0$ is initialized such that $\bar{\mathcal{L}}(0, j_1, j_2) = M/N$ for all $0 \le j_1, j_2 \le N-1$. Then, we repeat the following computation

$$\bar{\mathcal{L}}_0 \rightarrow \bar{\mathcal{L}}_1 \rightarrow \bar{\mathcal{L}}_2 \rightarrow \cdots \rightarrow \bar{\mathcal{L}}_\Delta \rightarrow \bar{\mathcal{L}}_0,$$

until the matrix $\bar{\mathcal{L}}_0$ becomes stable. Since we only need to keep $\bar{\mathcal{L}}_{\delta-1}$ when we calculate $\bar{\mathcal{L}}_\delta$, the number of variables is $O(N^2)$.

By using the above analytical method, the performance of the Canonical Algorithm in various static networks has been studied extensively. Closed form solutions to the expected load on each processor in dilation-1 embedding of complete trees and reproductions trees in completely connected networks and bus systems were found in [15]. In [16, 17, 28, 29], we evaluated the performance the Canonical Algorithm and its improved versions in hypercubes. Known results have also been reported for k-ary n-cubes and its special cases such as rings and tori [18]; asymmetric networks such as linear arrays [19] and meshes [5, 21, 25]; symmetric networks like barrel-shifters and Illiac networks [20], cube-connected cycles [22], and hypercubic networks, including wrapped butterflies, shuffle-exchange networks, and de Bruijn graphs [26].

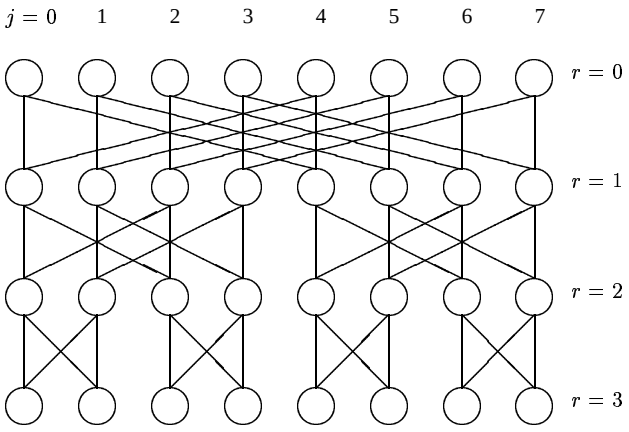

Figure 1. The 3-dimensional butterfly network.

4 Embedding in Butterfly Networks

In a c-dimensional *butterfly* having $N = (c+1)2^c$ processors, a processor is represented by $P_{(r,j)}$, where $0 \le r \le c$, and $0 \le j < 2^c$ (see Figure 1). Processors $P_{(r,0)}$, $P_{(r,1)}$, ..., $P_{(r,2^c-1)}$ comprise level r. Generally, processors in level r are connected to those in levels $r-1$ and $r+1$. In particular, $P_{(r,j)}$ has four neighbors $P_{(r\pm1,j)}$, $P_{(r-1,j')}$, $P_{(r+1,j'')}$, where $j = j_1 j_2 \cdots j_c$ and $j' = j'_1 j'_2 \cdots j'_c$ differ exactly in the rth bit, and j and $j'' = j''_1 j''_2 \cdots j''_c$ differ exactly in the $(r+1)$st bit, where $1 \le r \le c-1$, and $0 \le j < 2^c$. However, processors in level 0 are connected only to those in level 1, while processors in level c are connected only to those in level $c-1$. In particular, $P_{(0,j)}$ has two neighbors, i.e., $P_{(1,j)}$ and $P_{(1,j')}$, where j and j' differ exactly in

the first bit, and $P_{(c,j)}$ has two neighbors, i.e., $P_{(c-1,j)}$ and $P_{(c-1,j')}$, where j and j' differ exactly in the last bit. (See [14] for detailed discussion on the butterfly network and its variations.)

Let us define $L_B(h, \delta, r, j, r', j')$, where $h \geq 0$, $0 \leq \delta \leq \Delta$, $0 \leq r, r' \leq c$, $0 \leq j, j' < 2^c$, to be the expected load on processor $P_{(r',j')}$ in a dilation-Δ embedding (produced by the Canonical Algorithm) of a complete b-ary tree with height h in a butterfly network, when $P_{(r,j)}$ is the initial processor, and the root has δ more steps to go in its random walk. By using Theorem 1, we get a recurrence relation for $L_B(h, \delta, r, j, r', j')$ in Figure 2.

Similarly, let $\bar{L}_B(\delta, r, j, r', j')$ where $0 \leq \delta \leq \Delta$, $0 \leq r, r' \leq c$, $0 \leq j, j' < 2^c$, be the expected load on processor $P_{(r',j')}$ in a dilation-Δ embedding (produced by the Canonical Algorithm) of a reproduction tree in a butterfly network, when $P_{(r,j)}$ is the initial processor, and the root of the tree has δ more steps to go. By using Theorem 2, we get a recurrence relation for $\bar{L}_B(\delta, r, j, r', j')$ in Figure 3.

In the above recurrence relations, if $j = j_1 j_2 \cdots j_c$, then $j^{(r)} = j_1 j_2 \cdots \bar{j}_r \cdots j_c$, where $1 \leq r \leq c$.

5 Numerical Data

To show the performance data, we consider a 3-dimensional butterfly network with $N = 32$ processors. In Tables 1-3, we demonstrate the performance ratio α_h of the Canonical Algorithm in embedding complete b-ary trees.

Table 1 shows that the choice of the initial processor $P_{(r^*,j^*)}$ affects α_h. However, no matter what the initial processor is, eventually α_h converges to the same asymptotic performance ratio α_∞ as $h \to \infty$.

Table 2 shows the impact of dilation Δ, i.e., the length of a random walk. It is noticed that butterflies are bipartite networks. Therefore, if Δ is even, only half of the processors are used in embedding, which implies that $\alpha_h \geq 2$ [30]. To achieve good performance, Δ should be odd. For the same parity of Δ, larger Δ does improve the performance ratio α_h, and in such a case, as $h \to \infty$, α_h approaches the asymptotic performance ratio α_∞ which depends on the parity of Δ.

In Table 3, we show the performance ratio α_h for various branching factor b, and we observe that α_h is an increasing function of b. The asymptotic performance ratio α_∞ depends on b.

Based on the above numerical data, we conjecture that the asymptotic performance ratio of the Canonical Algorithm in embedding complete b-ary trees in a c-dimensional butterfly network is

$$\alpha_\infty = \left(\frac{2b}{b+1}\right)\left(1 + \frac{1}{c}\right), \quad \Delta \text{ odd},$$

and

$$\alpha_\infty = 2\left(1 + \frac{1}{c}\right), \quad \Delta \text{ even}.$$

When c is large, we have $\alpha_\infty \approx 2b/(b+1)$ for odd Δ and $\alpha_\infty \approx 2$ for even Δ. Such performance is comparable to that of the hypercubes [28].

$$
\begin{aligned}
&L_B(0,0,r,j,r,j) = 1, \quad 0 \leq r \leq c,\ 0 \leq j < 2^c;\\
&L_B(0,0,r,j,r',j') = 0,\\
&\quad 0 \leq r,r' \leq c,\ 0 \leq j,j' < 2^c,\ (r,j) \neq (r',j');\\
&L_B(h,0,r,j,r,j) = 1 + bL_B(h-1,\Delta,r,j,r,j),\\
&\quad h \geq 1,\ 0 \leq r \leq c,\ 0 \leq j < 2^c;\\
&L_B(h,0,r,j,r',j') = bL_B(h-1,\Delta,r,j,r',j'),\\
&\quad h \geq 1,\ 0 \leq r,r' \leq c,\ 0 \leq j,j' < 2^c,\ (r,j) \neq (r',j');\\
&L_B(h,\delta,r,j,r',j') =\\
&\begin{cases}
\frac{1}{4}\Big(L_B(h,\delta-1,r\pm1,j,r',j')\\
\qquad +L_B(h,\delta-1,r-1,j^{(r)},r',j')\\
\qquad +L_B(h,\delta-1,r+1,j^{(r+1)},r',j')\Big),\\
\qquad 1 \leq r \leq c-1,\ 0 \leq j < 2^c;\\
\frac{1}{2}\Big(L_B(h,\delta-1,1,j,r',j')\\
\qquad +L_B(h,\delta-1,1,j^{(1)},r',j')\Big),\\
\qquad r = 0,\ 0 \leq j < 2^c;\\
\frac{1}{2}\Big(L_B(h,\delta-1,c-1,j,r',j')\\
\qquad +L_B(h,\delta-1,c-1,j^{(c)},r',j')\Big),\\
\qquad r = c,\ 0 \leq j < 2^c;
\end{cases}\\
&\quad h \geq 0,\ 1 \leq \delta \leq \Delta,\ 0 \leq r,r' \leq c,\ 0 \leq j,j' < 2^c.
\end{aligned}
$$

Figure 2. Recurrence relation for the expected load $L_B(h, \delta, r, j, r', j')$.

$$
\begin{aligned}
&\bar{L}_B(0,r,j,r,j) = 1 + \bar{b}\bar{L}_B(\Delta,r,j,r,j),\\
&\quad 0 \leq r \leq c,\ 0 \leq j < 2^c;\\
&\bar{L}_B(0,r,j,r',j') = \bar{b}\bar{L}_B(\Delta,r,j,r',j'),\\
&\quad 0 \leq r,r' \leq c,\ 0 \leq j,j' < 2^c,\ (r,j) \neq (r',j');\\
&\bar{L}_B(\delta,r,j,r',j') =\\
&\begin{cases}
\frac{1}{4}\Big(\bar{L}_B(\delta-1,r\pm1,j,r',j')\\
\qquad +\bar{L}_B(\delta-1,r-1,j^{(r)},r',j')\\
\qquad +\bar{L}_B(\delta-1,r+1,j^{(r+1)},r',j')\Big),\\
\qquad 1 \leq r \leq c-1,\ 0 \leq j < 2^c;\\
\frac{1}{2}\Big(\bar{L}_B(\delta-1,1,j,r',j')\\
\qquad +\bar{L}_B(\delta-1,1,j^{(1)},r',j')\Big),\\
\qquad r = 0,\ 0 \leq j < 2^c;\\
\frac{1}{2}\Big(\bar{L}_B(\delta-1,c-1,j,r',j')\\
\qquad +\bar{L}_B(\delta-1,c-1,j^{(c)},r',j')\Big),\\
\qquad r = c,\ 0 \leq j < 2^c;
\end{cases}\\
&\quad 1 \leq \delta \leq \Delta,\ 0 \leq r,r' \leq c,\ 0 \leq j,j' < 2^c.
\end{aligned}
$$

Figure 3. Recurrence relation for the expected load $\bar{L}_B(\delta, r, j, r', j')$.

Table 1. Performance Ratio α_h vs. Initial Processor (r^*, j^*). ($N = 32$, complete tree with $b = 2$, $\Delta = 1$)

h	$(0,0)$	$(0,4)$	$(1,0)$	$(1,4)$	$(2,0)$	$(2,4)$
5	5.333	5.333	3.556	3.556	3.556	3.556
6	2.898	2.898	4.661	4.661	4.661	4.661
7	4.016	4.016	3.075	3.075	3.075	3.075
8	2.380	2.380	3.601	3.601	3.601	3.601
9	3.222	3.222	2.737	2.737	2.737	2.737
10	2.243	2.243	2.994	2.994	2.994	2.994
11	2.739	2.739	2.493	2.493	2.493	2.493
12	2.131	2.131	2.619	2.619	2.619	2.619
13	2.437	2.437	2.313	2.313	2.313	2.313
14	2.044	2.044	2.376	2.376	2.376	2.376
15	2.241	2.241	2.179	2.179	2.179	2.179
16	1.978	1.978	2.210	2.210	2.210	2.210
17	2.109	2.109	2.078	2.078	2.078	2.078
18	1.928	1.928	2.094	2.094	2.094	2.094
19	2.019	2.019	2.003	2.003	2.003	2.003
20	1.890	1.890	2.011	2.011	2.011	2.011
21	1.955	1.955	1.947	1.947	1.947	1.947
22	1.862	1.862	1.951	1.951	1.951	1.951
23	1.908	1.908	1.904	1.904	1.904	1.904
24	1.841	1.841	1.906	1.906	1.906	1.906
25	1.875	1.875	1.873	1.873	1.873	1.873
26	1.825	1.825	1.874	1.874	1.874	1.874
27	1.850	1.850	1.849	1.849	1.849	1.849
28	1.813	1.813	1.850	1.850	1.850	1.850
29	1.832	1.832	1.831	1.831	1.831	1.831
30	1.805	1.805	1.831	1.831	1.831	1.831

Table 2. Performance Ratio α_h vs. Random Walk Length Δ. ($N = 32$, complete tree with $b = 2$, $(r^*, j^*) = (0,0)$)

h	$\Delta = 1$	$\Delta = 2$	$\Delta = 3$	$\Delta = 4$	$\Delta = 5$	$\Delta = 6$
5	5.333	3.421	2.538	2.924	2.056	2.754
6	2.898	3.280	1.959	2.837	1.821	2.717
7	4.016	3.150	2.083	2.775	1.862	2.695
8	2.380	3.042	1.865	2.734	1.793	2.682
9	3.222	2.954	1.902	2.708	1.802	2.675
10	2.243	2.886	1.817	2.692	1.782	2.671
11	2.739	2.832	1.829	2.681	1.785	2.669
12	2.131	2.792	1.795	2.675	1.779	2.668
13	2.437	2.761	1.799	2.672	1.780	2.667
14	2.044	2.738	1.785	2.670	1.778	2.667
15	2.241	2.720	1.787	2.668	1.778	2.667
16	1.978	2.707	1.781	2.668	1.778	2.667
17	2.109	2.697	1.782	2.667	1.778	2.667
18	1.928	2.689	1.779	2.667	1.778	2.667
19	2.019	2.684	1.779	2.667	1.778	2.667
20	1.890	2.679	1.778	2.667	1.778	2.667
21	1.955	2.676	1.778	2.667	1.778	2.667
22	1.862	2.674	1.778	2.667	1.778	2.667
23	1.908	2.672	1.778	2.667	1.778	2.667
24	1.841	2.671	1.778	2.667	1.778	2.667
25	1.875	2.670	1.778	2.667	1.778	2.667
26	1.825	2.669	1.778	2.667	1.778	2.667
27	1.850	2.668	1.778	2.667	1.778	2.667
28	1.813	2.668	1.778	2.667	1.778	2.667
29	1.832	2.668	1.778	2.667	1.778	2.667
30	1.805	2.667	1.778	2.667	1.778	2.667

The performance of the Canonical Algorithm in embedding reproduction trees is shown in Table 4, where $\alpha_{\overline{M}} = L_{MAX}/(\overline{M}/N)$, and $\overline{M} = 1/(1-\bar{b})$ is the expected number of nodes in a reproduction tree. A general treatment of embedding reproduction trees is given in [24], where it is proven that for an arbitrary network $\mathcal{N}$ with N processors, as $\overline{M} \to \infty$, $\alpha_{\overline{M}}$ approaches the ratio of the maximum node degree to the average node degree, i.e.,

$$\lim_{\overline{M}\to\infty} \alpha_{\overline{M}} = N\left(\frac{\max(d_0, d_1, d_2..., d_{N-1})}{d_0 + d_1 + d_2 + \cdots + d_{N-1}}\right),$$

where $d_j = \deg(\mathcal{N}, j)$ is the degree of processor P_j. By the above result, we know that when Δ is odd, the asymptotic performance ratio of the Canonical Algorithm in embedding reproduction trees in butterflies is

$$\alpha_\infty = 1 + \frac{1}{c}.$$

When c is large, we have $\alpha_\infty \approx 1$ for odd Δ.

Table 3. Performance Ratio α_h vs. Branching Factor b. ($N = 32$, complete tree, $\Delta = 1$, $(r^*, j^*) = (0,0)$)

h	$b = 2$	$b = 3$	$b = 4$	$b = 5$	$b = 6$	$b = 7$
5	5.333	5.547	5.767	5.941	6.075	6.181
6	2.898	2.831	3.012	3.133	3.220	3.286
7	4.016	4.199	4.392	4.538	4.648	4.733
8	2.380	2.648	2.813	2.925	3.006	3.067
9	3.222	3.425	3.603	3.732	3.827	3.900
10	2.243	2.492	2.648	2.755	2.831	2.888
11	2.739	2.959	3.126	3.243	3.329	3.394
12	2.131	2.371	2.521	2.623	2.696	2.751
13	2.437	2.665	2.823	2.933	3.012	3.073
14	2.044	2.278	2.424	2.523	2.594	2.647
15	2.241	2.472	2.624	2.728	2.803	2.860
16	1.978	2.209	2.352	2.448	2.517	2.569
17	2.109	2.340	2.488	2.588	2.660	2.714
18	1.928	2.157	2.297	2.391	2.459	2.510
19	2.019	2.248	2.392	2.490	2.559	2.612
20	1.890	2.118	2.256	2.349	2.416	2.466
21	1.955	2.183	2.324	2.419	2.488	2.539
22	1.862	2.088	2.225	2.317	2.383	2.433
23	1.908	2.136	2.275	2.368	2.435	2.486
24	1.841	2.066	2.202	2.294	2.359	2.408
25	1.875	2.101	2.239	2.331	2.397	2.447
26	1.825	2.050	2.185	2.276	2.341	2.389
27	1.850	2.075	2.212	2.303	2.369	2.418
28	1.813	2.037	2.172	2.262	2.327	2.375
29	1.832	2.056	2.192	2.283	2.348	2.397
30	1.805	2.028	2.162	2.252	2.317	2.365

6 Conclusions

We have evaluated the performance of a random-walk-based algorithm for embedding dynamically evolving trees in butterfly networks. Our performance data demonstrate that butterflies have comparable performance with hypercubes in supporting tree-structured parallel computations.

Table 4. Performance Ratio $\alpha_{\overline{M}}$ vs. Random Walk Length Δ. ($N = 32$, reproduction tree, $(r^*, j^*) = (0, 0)$)

$\overline{M}$	$\Delta = 1$	$\Delta = 2$	$\Delta = 3$	$\Delta = 4$	$\Delta = 5$	$\Delta = 6$
100	1.649	2.708	1.426	2.672	1.377	2.659
200	1.495	2.688	1.380	2.669	1.355	2.663
300	1.442	2.681	1.365	2.669	1.348	2.664
400	1.415	2.678	1.357	2.668	1.344	2.665
500	1.399	2.675	1.352	2.668	1.342	2.665
600	1.388	2.674	1.349	2.668	1.341	2.665
700	1.380	2.673	1.347	2.667	1.340	2.666
800	1.374	2.672	1.345	2.667	1.339	2.666
900	1.370	2.672	1.344	2.667	1.338	2.666
1000	1.366	2.671	1.343	2.667	1.338	2.666
1100	1.363	2.671	1.342	2.667	1.337	2.666
1200	1.361	2.670	1.341	2.667	1.337	2.666
1300	1.359	2.670	1.341	2.667	1.337	2.666
1400	1.357	2.670	1.340	2.667	1.336	2.666
1500	1.355	2.670	1.340	2.667	1.336	2.666
1600	1.354	2.669	1.339	2.667	1.336	2.666
1700	1.353	2.669	1.339	2.667	1.336	2.666
1800	1.352	2.669	1.339	2.667	1.336	2.666
1900	1.351	2.669	1.338	2.667	1.336	2.666
2000	1.350	2.669	1.338	2.667	1.336	2.666
2100	1.349	2.669	1.338	2.667	1.335	2.666
2200	1.348	2.669	1.338	2.667	1.335	2.666
2300	1.348	2.669	1.337	2.667	1.335	2.666
2400	1.347	2.669	1.337	2.667	1.335	2.666
2500	1.347	2.668	1.337	2.667	1.335	2.666
∞	1.333	2.667	1.333	2.667	1.333	2.667

Acknowledgment

The research is supported by US National Science Foundation under grant CCR-0091719.

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.

[3] S. Bhatt and J.-Y. Cai, "Take a walk, grow a tree," *Proc. of IEEE Symp. on Foundations of Computer Science*, pp. 469-478, 1988.

[4] S. Bhatt, D. Greenberg, T. Leighton, and P. Liu, "Tight bounds for on-line tree embeddings," *SIAM Journal on Computing*, vol. 29, no. 2, pp. 474-491, 1999.

[5] C.-P. Cao, *Simulation and Performance Evaluation of Dynamic Tree Embedding on Meshes*, MS Thesis, Department of Mathematics and Computer Science, State University of New York, New Paltz, New York, May 1997.

[6] Ö. Eğecioğlu and M. Ibel, "Asymptotic hypercube embeddings of dynamic k-ary trees," *Congressus Numerantium*, vol. 126, pp. 21-32, 1997.

[7] W. Feller, *An Introduction to Probability Theory and Its Applications*, Vol. 1, 3rd Ed., John Wiley & Sons, 1968.

[8] J. Gaber and B. Toursel, "Randomized load distribution of arbitrary trees on a distributed network," *Proc. of 13th Annual ACM Symp. on Applied Computing*, pp. 564-568, Atlanta, Georgia, February 1998.

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

[10] V. Heun and E.W. Mayr, "Efficient dynamic embedding of arbitrary binary trees into hypercubes," *Lecture Notes in Computer Science*, vol. 1117, pp. 287-298, Springer-Verlag, 1996.

[11] 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.

[12] 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.

[13] 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.

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

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

[16] K. Li, "On dynamic tree growing in hypercubes," *Proc. of 12th Annual ACM Symp. on Applied Computing*, pp. 496-503, San Jose, California, February 1997.

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

[18] 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, Las Vegas, Nevada, June 1997.

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

[20] 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, Dayton, Ohio, July 1997.

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

[22] K. Li, "Performance evaluation of probabilistic tree embedding in cube-connected cycles," *Proc. of 13th Annual ACM Symp. on Applied Computing*, pp. 584-592, Atlanta, Georgia, February 1998.

[23] K. Li, "Asymptotically optimal randomized tree embedding in static networks," *Proc. of IPPS/SPDP '98* (merged symposium of *the 12th International Parallel Processing Symposium* and *the 9th Symposium on Parallel and Distributed Processing*), pp. 423-430, Orlando, Florida, March 1998.

[24] K. Li, "Analyzing the performance of a probabilistic algorithm for embedding reproduction trees in static networks," *Proc. of 6th High Performance Computing Symposium*, pp. 197-202, Boston, Massachusetts, April 1998.

[25] K. Li, "Asymptotic performance analysis of randomized tree growing in meshes," *Proc. of 6th High Performance Computing Symposium*, pp. 222-227, Boston, Massachusetts, April 1998.

[26] K. Li, "A comparative performance evaluation of randomized tree embedding in hypercubic networks," *Proc. of International Conference on Parallel and Distributed Processing Techniques and Applications*, vol. IV, pp. 1796-1805, Las Vegas, Nevada, July 1998.

[27] K. Li, "Efficient randomized load distribution for tree structured computations on parallel and distributed computer systems," *International Journal of Computer Mathematics*, vol. 71, pp. 21-34, 1999.

[28] K. Li, "A method for evaluating the expected load of dynamic tree embeddings in hypercubes," *International Journal on Foundations of Computer Science*, vol. 11, no. 2, pp. 207-230, 2000.

[29] K. Li and J.E. Dorband, "Asymptotically optimal probabilistic embedding algorithms for supporting tree structured computations in hypercubes," *Proc. of the 7th Symposium on the Frontiers of Massively Parallel Computations*, pp. 218-225, Annapolis, Maryland, February 1999.

[30] K. Li, Y. Pan, H. Shen, G.H. Young, and S.-Q. Zheng, "Lower bounds for dynamic tree embedding in bipartite graphs," *Journal of Parallel and Distributed Computing*, vol. 53, no. 2, pp. 119-143, 1998.

[31] K. Li and X. Shen, "Optimal embedding of healthy trees using random walk," *Proc. of 11th International Conference on Parallel and Distributed Computing and Systems*, vol. 1, pp. 50-55, Cambridge, Massachusetts, November 1999.

[32] U. Manber, *Introduction to Algorithms - A Creative Approach*, Addison-Wesley, 1989.

[33] B. Monien and H. Sudborough, "Embedding one interconnection network in another," in *Computational Graph Theory*, G. Tinhofer, E. Mayr, H. Noltemeier, and M.M. Syslo, eds., pp. 257-282, Springer-Verlag, New York, 1990.

[34] 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.

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

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

[37] 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," *Journal of Parallel and Distributed Computing*, vol. 50, no. 1, pp. 144-156, 1998.