Tight Bounds for On-line Tree Embeddings

Sandeep Bhatt[1] David Greenberg[2]
Tom Leighton[3] Pangfeng Liu[4]

[1] Bellcore, 445 South Street, Morristown NJ 07960.
[2] Sandia National Laboratories, Albuquerque NM 87185.
[3] Massachusetts Institute of Technology, Cambridge MA 02139.
[4] National Chung Cheng University, Chiayi 621, Taiwan.

Abstract

Tree-structured computations are relatively easy to process in parallel. As leaf processes are recursively spawned they can be assigned to independent processors in a multi-computer network. However, to achieve good performance the on-line mapping algorithm must maintain load balance, ie. distribute processes equitably among processors. Additionally, the algorithm itself must be distributed in nature, and process allocation must be completed via message-passing with minimal communication overhead.

This paper investigates bounds on the performance of deterministic and randomized algorithms for on-line tree embeddings. In particular, we study tradeoffs between computation overhead (load imbalance) and communication overhead (message congestion). We give a simple technique to derive lower bounds on the congestion that any on-line allocation algorithm must incur in order to guarantee load balance. This technique works for both randomized and deterministic algorithms. We prove that the advantage of randomization is limited. Optimal bounds are achieved for several networks including multi-dimensional grids and butterflies.

1 Introduction

Tree-structured computations arise naturally in diverse applications of the divide-and-conquer paradigm; for example, N-body simulations, the evaluation of functional expressions, backtrack searches, and branch-and-bound procedures. As the computation evolves, the corresponding tree grows and shrinks. Each node of the tree can recursively spawn subprocesses, and communicate with its parent. The most significant feature of tree computations is that each node can spawn children independently of and, therefore, in parallel with other nodes.

How do we exploit this inherent parallelism on a multicomputer network containing as few as two or as many as one hundred thousand processors? For maximum speedup, the processes must be evenly distributed

in the network. If neither the structure nor the size of the tree can be predicted at compile time, then a reasonable strategy might be to assign each newly spawned process to a randomly chosen processor. This gives even load distribution with high probability. Unfortunately, it also causes large congestion; many processes are spawned simultaneously and each sends startup information to its assigned processor over a long distance in the network. For any given network, one might reasonably suspect a trade-off between computational overhead (poor load distribution) and communication overhead (high congestion).

For the boolean hypercube and its derivative networks, e.g. the butterfly and cube-connected-cycle networks, simple and efficient algorithms have recently been developed and analyzed [1, 4]. These algorithms map processes to randomly chosen processors, but the random choice is confined to processors within a short distance from the spawning processor. In particular, every N-node tree can be dynamically spawned within the N-node hypercube so that, with high probability, each processor receives $O(1)$ tree nodes and such that the maximum congestion in the network is $O(1)$ [4]. Using this randomized tree embedding technique for butterfly, Ranade showed that optimal speedup can be achieved for backtrack search problems [6].

For the complete network Karp and Zhang showed that by assigning newly generated branch-and-bound tree nodes to random processors, the parallel branch-and-bound algorithm achieves linear speedup with high probability [3, 10]. The analysis in [3] was later simplified by Ranade [5].

In this paper we extend our study of on-line embeddings to additional multi-computer networks. We derive lower bounds on the congestion that any on-line, deterministic algorithm must incur in order to guarantee load balance. The bounds are tight, to within constant factors, for several networks including multi-dimensional grids and butterflies. The deterministic lower bound for butterflies contrasts sharply with previous randomized upper bounds [4]. We also adapt the techniques of [4] to develop a randomized algorithm for grids, and show that its performance is better than the deterministic lower bound. Finally, we give tight lower bounds on the expected congestion for randomized algorithms.

Our lower bounds are all based on the simpler problem of allocating tree nodes to two processors. It is well known that bounded-degree trees have small separators, and this seems to suggest that only a small amount of communication should be needed to distribute tree nodes evenly. Nevertheless, we show that, even in the simple case of two processors, large communication overhead is inevitable for balanced on-line embeddings.

The remainder of this paper is organized as follows. Section 2 defines the on-line embedding model and the cost measures studied in this paper. Section 3 presents lower bounds for deterministic algorithms. Section 4 discusses a deterministic off-line algorithm that embeds dynamic trees under a strict balance requirement. Section 5 presents randomized algorithms with improved performance. Section 6 presents lower bounds for randomized algorithms. Section 7 concludes the paper.

2 Models

We model the problem of growing trees on networks as an on-line embedding problem. Informally, the on-line tree embedding problem may be described as follows. We start with the root of a binary tree placed at some node of a multi-computer network. At any instant, each node with 0 or 1 child may choose to spawn a new child. Each newly spawned child must be assigned to a network node, and a path in the network must be chosen from the parent node to the child node. Furthermore, the decision of where to place the newly spawned child must be entirely local, i.e., the embedding algorithm can perform only local computations and no global data structures are allowed.

The on-line embedding algorithm is further constrained in that processes cannot migrate. Once embedded, a tree node cannot subsequently be moved to a different node in the network. These requirements are critical for fine-grain multi-computers [8]. Disallowing process migration is a common policy in practice; migration can conceivably lead to thrashing and cripple a large multi-computer network.

This framework for maintaining a dynamically evolving tree differs substantially from a model studied recently by Wu and Kung [9]. The latter model allows for a global data structure which all processors can access simultaneously. Moreover, newly spawned nodes do not need to be assigned processors immediately; the allocation can be deferred until some processor becomes idle. While the arguments and techniques used to prove tight upper and lower bounds are not affected drastically, the resulting bounds are substantially different.

2.1 Growth sequences

Although many tree nodes can spawn children simultaneously, it will suffice for our lower bound arguments to assume that the tree grows one node at a time. Formally, an instance of the on-line tree embedding problem is a sequence of directed edges, $e_i = (w_j, w_i)$, $1 \leq j < i$, in which w_i is the i-th node spawned and has parent w_j. Node w_1 is the root. A *growth sequence* is a sequence of edges such that, for each k, the set $\{e_i : i \leq k\}$ of edges forms a directed tree, with each node having outdegree at most two.

The restriction to binary trees is arbitrary, any bounded degree would do, but since the networks we consider have bounded degree, trees with unbounded fanout lead, uninterestingly, to poor worst-case performance.

2.2 Embeddings and their quality

An embedding of a tree T in a network H is a mapping of nodes of T to vertices of H, together with a mapping from edges of T to paths in H. The mapping of nodes is not required to be one-to-one; multiple processes can share the same processor. An on-line embedding induced by a growth sequence is a sequence of embeddings, in which each embedding is an extension of the previous one. This enforces the policy of disallowing process migration.

Two standard measures of the quality of an embedding are dilation and congestion. The *dilation* of an embedding is the maximum path length in the network corresponding to an edge of T. The *congestion* of an embedding is the maximum number of edges of T whose images traverse any edge in H.

An embedding of a k-node tree is said to be α-*balanced*, $\alpha \geq 1$, if no more than $\alpha\lceil k/P \rceil$ tree nodes are assigned to any of the P nodes of H. The overloading factor, α, is typically a small constant, independent of P, the size of the network.

In some tree-structured computations, the computational activity is limited to the leaves of the tree. In such cases, it is important only to evenly distribute the leaves within the network. An on-line embedding of a tree with ℓ leaves is said to be α-*balanced on the leaves*, $\alpha \geq 1$, if no more than $\alpha\lceil \ell/P \rceil$ leaves are assigned to any of the P processors of the network.

Finally, an on-line embedding for a growth sequence ρ is *continuously* α-*balanced*, if for every i, the i-th embedding (induced by the first i elements of ρ) is α-balanced. On the other hand, if only the final embedding in the sequence is α-balanced, then we say that the on-line embedding is *terminally* α-*balanced*. Obviously, terminal balance is a weaker requirement than continuous balance. Also note that in order to distinguish the two balance requirements under the on-line model, the length of the growth sequence will be given to the embedding algorithms.

3 Deterministic Lower Bounds

For a given embedding of a tree T, an edge $e = (u, v)$ in T is said to be a *cut edge* if the end points u and v of e are mapped to distinct processors. Each cut edge corresponds to a child spawned at a remote processor, thus requiring communication through the network. The number of cut edges can critically affect congestion.

3.1 Lower bound on two processor cut edges

Given a deterministic on-line algorithm we show how to construct an N-node binary tree for which the algorithm either requires $\Theta(N)$ cut edges or fails to achieve terminally α-balanced, for $\alpha < 2$. We describe an *adversary* which progressively grows a tree such that if the algorithm does not use many cut edges then the final embedding will not be balanced.

On-line allocation for two processors is equivalent to coloring each tree node either black or white. Without loss of generality, suppose that the root is colored *black*. The adversarial tree construction works in phases: in the first phase, we spawn two children from every black node. This phase ends when all the leaves are white. Phase 2 continues with this strategy of spawning from one color only, spawning two children from every white node until all the leaves are black. The tree is grown in this "layer by layer" method, producing leaves of alternating colors at the ends of successive phases.

Since the 2-coloring algorithm is deterministic, the color committed on-line to each newly spawned node is "known," i.e., computable by the adversary constructing the growth sequence. Thus for each possible deterministic algorithm the adversary can produce a well-defined tree.

We will show that the number of cut edges grows linearly with the size of the adversarial tree, under the requirement of terminal balance. The proof is based on the following observation. At the end of every phase each leaf is connected to its parent by a cut edge. Since every internal node has two children, more than half the edges of the tree (those incident to leaves) are cut edges.

We use the following terminology. A tree node is in layer i if the path from the root to the node contains $i-1$ cut edges. Let n_i be the number of nodes in layer i. Each node spawned during phase i lies either in layer i or in layer $i+1$.

Lemma 1 *For every deterministic algorithm $\mathcal{A}$, $\alpha < 2$, and $\epsilon < (2-\alpha)/4$ there exists a growth sequence ρ of length $N > \alpha/(2-\alpha-4\epsilon)$ such that if the on-line embedding of ρ is terminally α-balanced then the number of cut edges is greater than ϵN.*

Proof.

We consider two cases: either the embedding of ρ is completed before phase 1 terminates, or else the embedding is completed in a later phase.

The embedding is completed before phase 1 terminates. In this case every white node is a leaf and the number of cut edges equals the number of white leaves. In order to maintain terminal balance, there must be at least $N - \alpha\lceil N/2\rceil$ white nodes. This number is greater than $N-\alpha(N/2+1/2)$, which exceeds ϵN when $N > \alpha/(2-\alpha-4\epsilon)$. Therefore there are at least ϵN cut edges.

The embedding is completed during phase k, $k > 1$. Let n_i be the number of nodes in layer i. We again distinguish between two subcases: either $\sum_1^{k-1} n_i \geq \epsilon N$, or $\sum_1^{k-1} n_i < \epsilon N$.

When $\sum_1^{k-1} n_i \geq \epsilon N$.

We first observe that every node in layer $k-1$ is an internal node. Together with the fact that every internal node has two children, we conclude that there are $1 + \sum_1^{k-1} n_i$ cut edges, each connecting a node in layer $k-1$ to a node in layer k. Thus, in this subcase the number of cut edges is greater than ϵN.

When $\sum_1^{k-1} n_i < \epsilon N$.

In this subcase $n_k + n_{k+1} > N(1-\epsilon)$. From the terminal balance condition, we also have $n_k < \alpha\lceil N/2\rceil$. These two imply that $n_{k+1} > N(1-\epsilon) - \alpha\lceil N/2\rceil > N(1-\epsilon-\alpha/2) - \alpha/2 > \epsilon N$ when $N > \alpha/(2-4\epsilon-\alpha)$.

Since the embedding terminated during phase k, and before phase $k+1$ began, it follows that every node in layer $k+1$ is a leaf that is connected to its parent by a cut edge. From the preceding argument it again follows that the number of cut edges is greater than ϵN. ■

Lemma 1 implies that every deterministic, on-line, load-balancing algorithm requires a linear number of cut edges on some finite sequence. This lower bound for terminally balanced algorithms holds for continuously balanced algorithms as well.

3.2 Multiple processors

As we show in this section, Lemma 1 can be extended in a straightforward manner to any bounded number of processors. With P colors (processors) instead of two, we use a similar construction to the one in Lemma 1. The only difference is that each phase grows only one subtree, rooted at a leaf grown in the previous phase. Each phase ends when all the leaves grown during that phase are colored differently from the root of the subtree. We also use the same terminology. A tree node is in the layer i if the path from the node to the root contains $i-1$ cut edges. The symbol n_i denotes the number of nodes in layer i.

Lemma 2 *For every deterministic algorithm $\mathcal{A}$, $P \geq 2$, $\alpha < P$, and $\epsilon < \frac{1}{2}(1-\alpha/P)$ there exists a growth sequence ρ of length $N > \alpha/(1-2\epsilon-\alpha/P)$ such that if the on-line embedding of ρ is terminally α-balanced among P processors then the number of cut edges is greater than ϵN.*

Proof.

We consider two cases: either the construction of ρ ends in the first phase or in a later phase.

The construction ends in the first phase. In this case every node in the second layer has a cut-edge to its parent. In order to maintain terminal α-balance, the number of nodes in the second layer must be at least $N - \alpha\lceil N/P \rceil$. Therefore the number of cut edges is at least $N - \alpha\lceil N/P \rceil > N - \alpha(N/P + 1) > \epsilon N$ when $N > \alpha/(1 - 2\epsilon - \frac{\alpha}{P})$.

The construction ends during the k-th phase for $k > 1$. Let T_k be the subtree constructed during the k-th phase and $r \in T_k$ be the root of T_k. T_k is partitioned into two subsets, L and R. L is the set of nodes which are in layer $k+1$ and $R = T_k - L$. Notice that L contains only leaves of T_k since ρ ends in phase k.

Every node in the tree induced by $\rho - T_k$ has 0 or 2 children (except the parent of r), hence the number of leaves in $\rho - T_k$ is $\frac{1}{2}(N - |T_k|)$. Additionally, every *leaf* in $(\rho - T_k) \cup L$ has a cut edge to its parent, so the number of cut edges is at least $\frac{1}{2}(N - |L| - |R|) + |L| \geq \frac{1}{2}(N - |R|)$. Therefore, by bounding the size of R we get a lower bound on the number of cut edges. The size of R is at most $\alpha\lceil \frac{N}{P} \rceil$ because it consists of only one color (the color of r). Therefore the number of cut edges is at least $\frac{1}{2}(N - |R|) \geq \frac{1}{2}(N - \alpha\lceil \frac{N}{P} \rceil) \geq \frac{1}{2}(N - \alpha(\frac{N}{P} + 1)) \geq \frac{N}{2}(1 - \alpha/P) - \frac{\alpha}{2} > \epsilon N$ when $N > \alpha/(1 - \frac{\alpha}{P} - 2\epsilon)$. ■

As mentioned in the previous section, in some tree structured computations, it is important to evenly distribute only the leaves of the tree. We next generalize Lemma 2 to get a lower bound on the number of cut-edges when only the leaves need to be terminally balanced.

Lemma 3 *For every deterministic algorithm $\mathcal{A}$, $P \geq 2$, $\alpha < P$ and $\epsilon < \frac{1}{2}(1 - \frac{\alpha}{2P})$ there exists a growth sequence ρ of length $N > 2\alpha/(1 - \frac{\alpha}{P} - 2\epsilon)$ such that if the on-line embedding of ρ is terminally α-balanced on the leaves then the number of cut edges is greater than ϵN.*

Proof.

We use the adversarial strategy in Lemma 2 to grow a tree with N nodes, $N > 2\alpha/(1 - \frac{\alpha}{P} - 2\epsilon)$. In order to simplify the calculation, we assume, without loss of generality, that N is even, so that there are exactly $N/2$ leaves in the N-node tree.

In the N-node tree grown by the adversary, let S denote the set of leaves which have the same color as their parents. Every leaf that has the same color as its parent must have been spawned in the last phase. Every leaf not in S is connected to its parent by a cut-edge, so that the number of cut edges is at least $N/2 - |S|$.

Since all leaves in S are grown in the same phase, they have the same color as the root of the subtree that is grown during that phase. In order to satisfy the terminal α-balance requirement on the leaves, the size of S is at most $\alpha\lceil\frac{N}{2P}\rceil$. Hence the number of cut edges is at least $N/2 - |S| \geq N/2 - \alpha(\frac{N}{2P} + 1) \geq N(\frac{1}{2} - \frac{\alpha}{2P}) - \alpha \geq \epsilon N$ when $N > 2\alpha/(1 - \frac{\alpha}{P} - 2\epsilon)$. ■

3.3 Lower bounds on network congestion and dilation

The lower bound of Lemma 2 on the number of cut-edges required by any deterministic algorithm for P processors can be used to derive lower bounds on the worst-case congestion and dilation in different networks. In this section we derive tight lower bounds for multi-dimensional grids and butterfly networks.

We partition the given P-processor network into B blocks of equal size. We refer to an edge that connects processors in different blocks as a *cross-link*. An α-balanced embedding of an N node tree assigns at most $\alpha N/B$ tree nodes per block; for convenience we let N be a multiple of P so that all divisions are exact. For any specified α, we choose B such that $\alpha/B < 1$, so that each block in the partition can receive at most a small fraction of the tree nodes.

By Lemma 2, in the worst case, $\Omega(N)$ edges of the tree must be cut. If there are C cross-links in the network partition, then some cross-link must accommodate $\Omega(N/C)$ cut edges of the tree, causing congestion $\Omega(N/C)$. Observe that an upper bound on C yields a lower bound on congestion.

By removing the edges in every $\sqrt{P}/B$-th column, the P-node two-dimensional grid can be partitioned into B blocks of equal size with $B\sqrt{P}$ cross links. Consequently, the congestion is $\Omega(N/\sqrt{P})$. For the P-node butterfly $C = O(P/\log P)$ when the edges at every $O((\log P)/B)$-th level are removed; therefore, the congestion is $\Omega(\frac{N \log P}{P})$.

It is equally easy to obtain lower bounds on dilation. For a partition of the network into blocks, define the *boundary* of a block to be the set of processors incident to cross-links. Furthermore, let n_d be the number of processors within distance d from the boundary. Observe that, in any

embedding with dilation d, every tree node incident to a cut edge must lie within distance d from a boundary node of some block. Since there are $\Omega(N)$ cut-edges in the worst case, the total number of network nodes within distance d of boundary nodes must be large enough to accommodate $\Omega(N)$ tree nodes.

By removing the edges in every $\sqrt{P}/B$-th column, the P-node two-dimensional grid can be partitioned into B blocks of equal size, so that $n_d = \Theta(d\sqrt{P})$ for each block. From the balance constraint each processor has at most $\alpha\lceil N/P \rceil$ tree nodes. In order to accommodate $\Omega(N)$ tree nodes, it follows that $d = \Omega(\sqrt{P})$. Similarly, by removing the edges at every $O((\log P)/B)$-th level, the P-node butterfly can be partitioned so that $n_d = \Theta(dP/\log P)$. It follows that, for the butterfly, $d = \Omega(\log P)$.

These lower bounds can be matched by determinitic algorithms which map tree nodes in round-robin manner to processors so that the tree nodes are evenly distributed. First we observe that the lower bounds on dilation is alredy a constant factor of the diameters of the corresponding networks, therefore the dilation upper bounds immediately follow. Next we choose the route that connects adjacent tree nodes. For the two-dimensional grids the algorithm picks the direct route from a parent tree node to its children that makes at most one turn in the grid. As a result a tree edge whose image goes through a horizontal network link will have at least one endpoint embedded in the same row of processors as the communication link. Since the number of tree nodes embedded into a row of processors is $O(N/\sqrt{P})$ from the balance requirement, we conclude that the congestion on every horizontal link is $O(N/\sqrt{P})$. A similar argument holds for vertical links and the congestion bound follows. For butterfly networks we use standard off-line routing algorithms to meet the $O(\frac{N \log P}{P})$ congestion bounds.

The lower bounds on congestion and dilation for two-dimensional grids above extend in a straightforward manner to grids with multiple dimensions. We summarize our observations in the following theorem.

Theorem 1 *For every deterministic algorithm $\mathcal{A}$, and constant $\alpha > 1$, there exists a growth sequence ρ of length N (N sufficiently large) such that if the on-line embedding of ρ is terminally α-balanced, then (1) on the P-node, k-dimensional grid (constant k) the dilation is $\Omega(P^{1/k})$ and the congestion is $\Omega(N/P^{1-1/k})$, and (2) on the P-node butterfly the dilation is $\Omega(\log P)$ and the congestion $\Omega(\frac{N \log P}{P})$.*

3.4 Infinite growth sequences

In the previous sections we have shown that every deterministic, balanced (either terminally or continuously) algorithm is required to make $\Omega(N)$ cuts, on some growth sequences of sufficiently large length N. Turning the question around, one is led to ask: is there a growth sequence which is universally bad in the sense that every balanced algorithm is forced to make linearly many cuts?

As we will see in the next section, for every length-N growth sequence there is an *off-line* algorithm which is continuously α-balanced between two processors for $1 < \alpha < 2$, and requires only $O(\log^2 N)$ cut-edges.

However, in this section we show the existence of an infinite growth sequence on which every *on-line* continuously balanced algorithm requires, infinitely often, linearly many cut edges. Formally, we establish the following theorem.

Theorem 2 *There exists an infinite growth sequence ϕ^* such that, for every overloading factor $1 < \alpha < 2$, every on-line deterministic algorithm $\mathcal{A}$ which produces a continuously α-balanced embedding on two processors will, on infinitely many prefixes $\phi_1, \phi_2, \ldots$ of lengths $N_1, N_2, \ldots$ make at least ϵN_i cuts, with $\epsilon = \alpha(2-\alpha)/4(2+\alpha)$.*

Before proceeding to the proof of the above theorem, we explain the idea behind the proof. The infinite sequence is constructed using a standard diagonalization argument that invokes Lemma 1 repeatedly. We begin with an enumeration, $(\mathcal{A}_1, \mathcal{A}_2, \ldots)$ of all algorithms as a sequence in which every algorithm appears infinitely often.

The universal growth sequence is constructed in stages. The portion constructuted in phase i is denoted ϕ_i, and the entire growth sequence at the end of phase i is denoted $\mathcal{G}_i$, so that $\mathcal{G}_0$ consists of the root of the tree, and $\mathcal{G}_i = \mathcal{G}_{i-1} \circ \phi_i$ for $i > 0$ (the symbol $\circ$ denotes concatenation). The idea is to construct ϕ_i such that algorithm $\mathcal{A}_i$ requires linearly many cuts on $\mathcal{G}_i$. Since the number of cuts made by $\mathcal{A}_i$ on the initial prefix $\mathcal{G}_{i-1}$ can be very small, we make ϕ_i large enough so that the number of cut-edges forced in ϕ_i is a fraction of the length of $\mathcal{G}_i$. This latter idea is formalized in the following lemma.

Lemma 4 *Suppose that $\mathcal{A}$ is a continuously α-balanced, deterministic algorithm ($\alpha < 2$), and that ϕ is a growth sequence of length n. Then, for every $N > 2\alpha n/(2-\alpha)$, there is a growth sequence $\phi \circ \rho$ of length $n+N$ on which $\mathcal{A}$ makes at least $\epsilon(N+n)$ cut-edges, where $\epsilon = \alpha(2-\alpha)/4(2+\alpha)$.*

Proof.

For ease of exposition, we assume that $N+n$ is even. We use Lemma 1 to grow the sequence ρ of length N from a leaf of ϕ. We define layers on the subtree induced by ρ as in Lemma 1 and use n_i to denote the number of nodes in layer i of ρ.

We consider two cases: either the embedding of ρ is complete before phase 1 terminates, or else the embedding is completed during phase k, $k > 1$.

The embedding is complete before phase 1 terminates. In this case, every node in the second layer is incident to a cut edge so the number of cut edges in ρ equals n_2. In order to maintain α-balance, the number of nodes n_2 must be at least $N - \alpha\lceil\frac{N+n}{2}\rceil$. Therefore, the number of cut edges is at least $N - \alpha(\frac{N+n}{2})$. Since $N > 2\alpha n/(2-\alpha)$, we have that $N > 2\alpha(N+n)/(2+\alpha)$, so that the number of cut edges is greater than $\alpha(2-\alpha)(N+n)/2(2+\alpha) = 2\epsilon(N+n)$.

The embedding is complete after phase 1 terminates. We further divide the second case into two subcases, depending on whether $\sum_1^{k-1} n_i \geq \epsilon(N+n)$, or $\sum_1^{k-1} n_i < \epsilon(N+n)$.

When $\sum_1^{k-1} n_i \geq \epsilon(N+n)$

In this subcase, every node in the first $k-1$ layers is an internal node and has two children, so the number of cut edges between layers k and $k-1$ equals $\sum_1^{k-1} n_i + 1$, which exceeds $\epsilon(N+n)$.

When $\sum_1^{k-1} n_i < \epsilon(N+n)$

In this subcase $n_k + n_{k+1} > N - \epsilon(N+n)$. From the balance condition we have that $n_k \leq \alpha\lceil\frac{N+n}{2}\rceil$, so that $n_{k+1} > N - \epsilon(N+n) - \frac{\alpha}{2}(N+n)$. Using the inequality $N > 2\alpha(N+n)/(2+\alpha)$, it follows that $n_{k+1} > (\frac{2\alpha}{2+\alpha} - \epsilon - \frac{\alpha}{2})(N+n)$, which exceeds $\epsilon(N+n)$. ■

Proof of Theorem 2 Theorem 2 follows from applying Lemma 4 in the diagonalization argument outlined earlier, with $\epsilon = \alpha(2-\alpha)/4(2+\alpha)$. ■

4 Deterministic Algorithm for Off-line Model

In the off-line model the entire growth sequence is given in advance. The off-line embedding algorithm can preprocess this sequence to produce a sequence of embeddings. Since bounded-degree trees have small separators, terminal balance is easy to guarantee between two processors. The more interesting case is when the off-line algorithm is required to be continuously balanced.

Our off-line upper bound contrasts sharply with the on-line lower bound. An N-node binary tree can be partitioned into two subsets, each containing at least $\lfloor N/3 \rfloor$ nodes, by removing a single edge. Therefore an embedding algorithm can, by making a single cut-edge, guarantee terminal $\frac{4}{3}$-balance between two processors. Of course, this is possible only if each edge has a distinct identity which can be recognized by the algorithm when the child node adjacent to the edge is spawn.

Less trivially, we can show that, for every α, every length-N growth sequence can be *continuously* α-balanced between two processors with $O(\log^2 N)$ cuts, for any $1 < \alpha < 2$. This is achieved using multi-color bisectors [2].

Lemma 5 *([2]) The nodes of every N-node binary tree, each of whose nodes has one of k distinct colors, can be bisected into two equal size (to within one) subsets by removing at most $k \log N$ edges. Furthermore, for each of the k colors, the set of all nodes of the same color are divided equally (to within one) between the two subsets in the bisection.*

The algorithm first partitions the input growth sequence ρ into groups of consecutively spawned nodes. The partition $\{V_i\}$, with V_i of size N_i is determined as follows: The first N_1 nodes in the growth sequence are placed in the first partition V_1. The number N_1 is chosen to be an even number greater than $2/\gamma$, where $\gamma = \frac{2\alpha-2}{2-\alpha}$. The second group V_2 contains the next N_2 nodes; N_2 is chosen to be the largest even number no greater than γN_1. In general, V_i contains N_i tree nodes where N_i is the largest even number no greater than $\gamma \sum_{j=1}^{i-1} N_j$.

Once the partition is formed, the off-line algorithm applies Lemma 5 to the input tree in which every node in V_i has color i. When the tree is processed on-line, edge-by-edge, the new child is allocated depending on whether or not the current edge lies in the separator set S; if it does, the child is placed in the remote processor, otherwise it is placed in the same processor as the parent.

We now show that this algorithm maintains continuous α-balance. We first bound the number of groups in the partition. To this end, define a sequence, $\{M_i\}$, such that $M_1 = N_1$ and $M_i = \gamma \sum_{j=1}^{i-1} M_j$ for all $i \geq 2$.

Lemma 6 *$M_i = \gamma N_1 (1+\gamma)^{i-2}$ for $i \geq 2$.*

Proof.

By induction on i. Since $M_2 = \gamma N_1$, the basis, $i = 2$ is established. For the inductive step,

$$\begin{aligned} M_{i+1} &= \gamma \sum_{j=1}^{i} M_j \\ &= \gamma \sum_{j=1}^{i-1} M_j + \gamma M_i \\ &= M_i + \gamma M_i \\ &= (1+\gamma) M_i \\ &= \gamma N_1 (1+\gamma)^{i-1} \end{aligned}$$

■

Since, for every $i \geq 2$, N_i equals the largest even integer no greater than M_i, we can bound each N_i from below as follows.

Lemma 7 *$N_i > M_i - 2(1+\gamma)^{i-2}$ for $i \geq 2$.*

Proof.

By induction on i. Again, $N_2 > M_2 - 2$ by definition, so the basis is established. For the inductive step,

$$\begin{aligned} N_{i+1} &> \gamma \sum_{j=1}^{i} N_j \; - \; 2 \\ &> \gamma \sum_{j=1}^{i} M_j - \gamma \sum_{j=2}^{i} 2(1+\gamma)^{j-2}) \; - \; 2 \\ &= \gamma \sum_{j=1}^{i} M_j - 2\gamma \sum_{j=0}^{i-2} (1+\gamma)^j \; - \; 2 \\ &= M_{i+1} - 2(1+\gamma)^{i-1} \end{aligned}$$

■

Lemma 8 *The above off-line algorithm partitions any length N grow sequence into $O(\log N)$ groups, and produces $O(\log^2 N)$ cut edges.*

Proof.

From Lemmas 6 and 7, we conclude that $N_i > (\gamma N_1 - 2)(1+\gamma)^{i-2}$ for all $i \geq 2$. With $N_1 > 2/\gamma$, it follows that the sequence $\{N_i\}$ increases exponentially, so the number of groups in the partition is bounded by $O(\log N)$. And since the number of colors is bounded by $O(\log N)$, the resulting bisector S contains $O(\log^2 N)$ edges from Lemma 5. ■

Now we formally establish the continuously balancing property for the above off-line algorithm in the following theorem.

Theorem 3 *For every growth sequence ρ of length N, and $1 < \alpha < 2$, the off-line algorithm requires at most $O(\log^2 N)$ cut-edges and is continuously α-balanced between two processors for all prefixes of ρ of length $M > N_1$. (Where N_1 is as defined above.)*

Proof.

The number of cut-edges is bounded in Lemma 8. Since exactly half of the tree nodes in every V_i are assigned to each processor, the numbers of nodes in both processor are always equal at the end of every V_i, therefore we only need to show that the algorithm gives a continuously α-balanced embedding while processing every V_i. First, every V_j, $1 \leq j \leq i-1$, is perfectly balanced, so the number of white nodes equals the number of black nodes just before V_i is processed. Second, the number of white nodes in V_i equals the number of black nodes. Independent of the order in which the white and black nodes within V_i appear, we claim that α-balance is always guaranteed. To see this, consider the extreme case in which all the black nodes appear before the white nodes. The total number of black nodes after half the nodes in V_i appear equals $\frac{1}{2}(\sum_{j=1}^{i-1} N_j) + \frac{1}{2}N_i$. We claim that this is no more than the quantity allowed under α-balance which equals $\frac{\alpha}{2}(\sum_{j=1}^{i-1} N_j + N_i/2)$. To see this, consider the difference

$$\begin{aligned}
& \frac{\alpha}{2}\left(\sum_{j=1}^{i-1} N_j + N_i/2\right) - \frac{1}{2}(\sum_{j=1}^{i-1} N_j) - \frac{1}{2}N_i \\
= \;& \frac{\alpha-1}{2}\sum_{j=1}^{i-1} N_j \; - \; \frac{1-\alpha/2}{2}N_i \\
\geq \;& \frac{\alpha-1}{2}\sum_{j=1}^{i-1} N_j \; - \; \frac{2-\alpha}{4}\cdot\frac{2\alpha-2}{2-\alpha}\sum_{j=1}^{i-1} N_j \\
= \;& 0
\end{aligned}$$

■

5 Randomized Algorithms

5.1 n-way Balancing

The key tool for our randomized algorithms is the *n-way balancing transformation* [4] which evenly distributes a binary tree of size N into a ring

of $n = \log N$ processors. The n-way balancing transformation works as follows. A tree node is *distinguished* if it is in level $i \equiv 0 \bmod n/3$. For each distinguished node v we pick a random number $S(v)$ between 0 and $n/3$ as the *stretch count*. The transformation inserts a single dummy node in each edge in the subtree of height $S(v)$ rooted at v. The resulting tree after this transformation is denoted by $B(T)$. Define *level set* i to be the set of all tree nodes in a level congruent to i modulo n. Let the processors of the ring be p_0, $p_1 \cdots$ p_{n-1} in clockwise order. The algorithm embeds the tree nodes in level set i of $B(T)$ into processor p_i. Figure 1 illustrates an example of 6-way balancing.

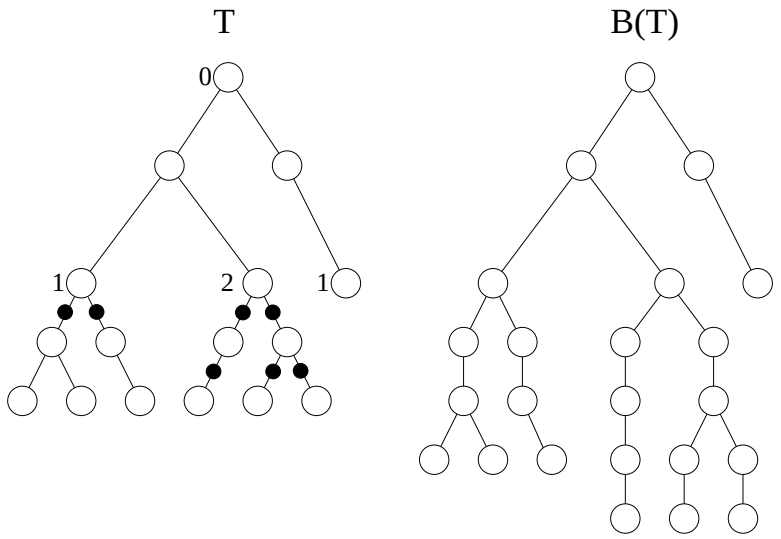

Figure 1: A 6-way balancing example taken from [6]. The solid circles indicate where the dummy nodes will be inserted, and the numbers next to the distinguished nodes are the stretch counts.

This transformation embeds every sufficiently large binary tree into a ring evenly with high probability, and with dilation two since at most one dummy node will be inserted into each edge in T.

Lemma 9 ([4]) *With probability* $1 - N^{-c}$, $c > 0$, *the above* $\log N$*-way balancing algorithm dynamically embeds every* N*-node tree into the* $\log N$*-node ring so that* $O(N/\log N)$ *tree nodes are mapped to any ring node.*

When the number of processors in the ring is less than $\log N$, we instead map a virtual $\log N$-length ring onto our smaller ring. In particular, with two processors, the first half of the virtual ring is mapped to one node and the second half to the other. Since the dilation of the mapping on the virtual ring is two, only the eight nodes of the virtual ring within distance two of the breaks contribute to the cut edges. Since these eight nodes contain at most $O(N/\log N)$ tree nodes, the number of cut edges is bounded by $O(N/\log N)$.

When the number of processors in the ring exceeds $\log N$, we divide the ring into $\log N$ groups, each containing an equal number of consecutive processors. We use $\log N$-way balancing to assign tree nodes to groups, and a deterministic strategy is used within each group to distribute nodes evenly in a group. For the P-node ring, the dilation is bounded by $O(P/\log N)$ because each tree edge traverse at most three groups. The bound on dilation is optimal; any mapping of the N-node complete binary tree on the P-node ring requires dilation $\Omega(P/\log N)$. From the dilation result, for any link l in the ring, only those processors that are within distance $O(P/\log N)$ can contribute congestion on l. Each processor can have at most $O(N/P)$ tree nodes in a balanced embedding and each tree node has at most three edges, therefore the congestion is $O(N/\log N)$ for any link in the ring. The congestion is a factor of $\log N$ less than the worst-case lower bound for deterministic on-line algorithms and matches the lower bound, shown in Section 6, on the expected value of the congestion for any randomized algorithms.

5.2 Cut-edges reduction

Suppose that we wish to equitably allocate the tree nodes on-line among P processors, with the goal of minimizing the total number of cut-edges. When P is any fixed constant, the strategy mentioned above uses $O(N/\log N)$ cuts. However, when P is $\log N$, each of the $N-1$ edges is cut. By modifying the $\log N$-way balancing technique slightly, we can reduce the total number of cuts to $O(N/\log \frac{N}{P})$. As we will see in the next section, this bound is the best possible.

We modify the algorithm so that we choose ϵ so $1 > \epsilon > 0$ and for each node in every $\epsilon \log \frac{N}{P}$-th level set of the transformed tree, we map it to a processor chosen uniformly at random among the P processors. We call these nodes *leaders*. Those nodes not in these level sets are embedded into the processor where their parents are embedded. We show that with high probability this algorithm gives balanced embedding and optimal number of cut edges when $P = O(N/\log^{\frac{1}{1-\epsilon}} N)$.

We need the following lemma [4] which states that, with high probability, the sum of independent random variables is at most a constant times their expected sum.

Lemma 10 ([4]) *Let $x_1, .., x_m$ be independent random variables in the range from 0 to V with $E(x_i) = \mu_i$, Let $X = \sum x_i$ and let $\mu = \sum_{i=1}^{m} \mu_i = E(X)$. Then for any constant β, $Pr(X \geq \beta\mu) \leq exp(-\beta\frac{\mu}{V})$.*

Theorem 4 *With probability $1 - N^{-c}$, $c > 0$, the above algorithm dynamically embeds every N-node binary tree into P processors where $P = O(N/\log^{\frac{1}{1-\epsilon}} N)$ so that $O(N/P)$ tree nodes are mapped to any processor and the number of cut edges is $O(N/\log \frac{N}{P})$.*

Proof.

First we estimate the number of cut edges. Since only edges at every $\epsilon \log \frac{N}{P}$-th level of $B(T)$ are cut, from Lemma 9 the total number of cut edges is $O(N/\log \frac{N}{P})$. This bound matches the lower bound on the expected number of cut edges, as we will see in the next section.

Let $l = \epsilon \log \frac{N}{P}$. The transformed tree is divided into m subtrees, $t_1, \ldots, t_m$, by cutting the edges between levels kl and $kl-1$ (for any integral k.) Each t_i is rooted at a leader and every node in t_i will follow this leader to a processor. Let M_i be the number of nodes in t_i and $x_{i,p}$ be the number of tree nodes in t_i that will be mapped into a particular processor p. We denote the total number of tree nodes assigned to p by $X_p = \sum_{i=1}^{m} x_{i,p}$.

The contributions from each subtree to any processor are independent and occur with equal probability. Therefore for all i, $1 \leq i \leq m$, $E(x_{i,p}) = M_i/P$ and $E(X_p) = N/P$. Moreover, all $x_{i,p}$'s are mutually independent and, since each tree contains at most the number of nodes in a complete binary tree of height $\epsilon \log \frac{N}{P}$, all $x_{i,p} \leq 2^{\epsilon \log \frac{N}{P}} = \frac{N}{P}^{\epsilon}$. Applying Lemma 10 we get:

$$\begin{aligned} Pr(X \geq \alpha \frac{N}{P}) &= Pr(X \geq \alpha E(X)) \\ &\leq exp(-\alpha \frac{(\frac{N}{P})}{(\frac{N}{P})^{\epsilon}}) \\ &= exp(-\alpha (N/P)^{1-\epsilon}) \\ &\leq exp(-c'(\log^{\frac{1}{1-\epsilon}} N)^{1-\epsilon}) \\ &< N^{-c'} \end{aligned}$$

With probability $1 - PN^{-c'} \geq 1 - N^{-c}$, every processor receives $O(N/P)$ tree nodes when $P = O(N/\log^{\frac{1}{1-\epsilon}} N)$. For larger values of P, the same bound is achievable, but the algorithm becomes more complicated and requires global information. ■

5.3 Two-dimensional grids

Any embedding of the N-node complete binary tree in the P-node two-dimensional grid, with at most $O(N/P)$ tree nodes per grid node, requires dilation $\Omega(\sqrt{P}/\log N)$ [7]. This follows from the fact that in any such embedding, some pair of tree nodes must be mapped distance $\Omega(\sqrt{P})$ apart, while the distance in the tree between the two nodes is $O(\log N)$. This lower bound on dilation is tight for off-line embeddings. In the previous section we saw that every online deterministic algorithm that balances the load requires dilation $\Omega(\sqrt{P})$ in the worst case. In this section we present a randomized algorithm which achieves the $O(\sqrt{P}/\log N)$ bound.

We shall see in Section 6 that the expected value of the congestion is $\Omega(\frac{N}{\sqrt{P}\log N})$. The randomized algorithm presented here will meet this bound.

5.3.1 Randomized algorithm for 2D-torus

For convenience, we work with the $2D$-torus instead of the grid. Since the $2D$-torus can be embedded efficiently within the grid, our bounds for the torus are tight, to within a constant factor, for the grid.

Our randomized algorithm for the torus extends the previous algorithm for rings. We partition the P-node torus into $\log^2 N$ square blocks, each of size $\frac{\sqrt{P}}{\log N}$ by $\frac{\sqrt{P}}{\log N}$. The block in row i and column j is denoted by B_{ij}. We use the $(\log N)$-way balancing transformation twice, independently for each dimension, to obtain two trees T_r and T_c. The random stretch counts in the two trees are chosen independently. Every node that is in level set i of T_r and level set j of T_c is mapped to B_{ij}. Within a block the tree nodes are distributed evenly in a deterministic manner.

Lemma 11 *With probability $1 - N^{-c}$, $c > 0$, the above algorithm dynamically embeds every N-node tree T in the $\sqrt{P} \times \sqrt{P}$ grid so that each block column receives $O(\frac{N}{\log N})$ tree nodes.*

Proof. Direct result from Lemma 9. ■

After knowing that each column receives $O(N/\log N)$ nodes, we show that within a column, the distribution to a particular block is the sum of many mutually independent random variables. Then by using Lemma 10 we can show that the algorithm distributes nodes evenly among the blocks in a column.

Lemma 12 *Given that each column receives $O(N/\log N)$ tree nodes, with probability $1 - N^{-c}$, $c > 0$, the above algorithm dynamically embeds every N-node tree T in the $\sqrt{P} \times \sqrt{P}$ grid so that each block receives $O(\frac{N}{\log^2 N})$ tree nodes.*

Proof.

The analysis and terminology follows the analysis and terminology in [4] quite closely.

We will examine how tree nodes are distributed among the blocks in one column. Let column i be the block column with largest number of tree nodes, I be the set of nodes from T that are in block column i and call any node in I an i-node. Note that all the i-nodes are in the level set i of T_c.

Recall that $n = \log N$. Tree T is divided into three zones. Zone 0 contains nodes that are in level sets 0 through $n/3 - 1$, zone 1 contains level sets $n/3$ through $2n/3 - 1$, and zone 3 contains the rest. Define the *triple* for the i-th level set as the level sets i, $i + n/3$ and $i + 2n/3$.

Zone 1 of T is naturally partitioned into a set of forests $f_1, \ldots, f_m$. Each forest consists of all the trees in zone 1 that have the same nearest common ancestors at level set 0 (that is the top of zone 0). Let these ancestors be $r_1, \ldots, r_m$ respectively. For each forest f_j, $1 \leq j \leq m$, let M_j be the number of i-nodes in f_j and $x_{j,q}$ be the number of i-nodes that are mapped into the triple of level set q.

We want to show that I is evenly distributed among all the level sets of T_r. Since the number of nodes in any level set is bounded by the number of nodes in its triple, it suffices to show that with high probability, any triple receives at most $O(\frac{N}{\log^2 N})$ tree nodes from I.

We show that the distribution from I to the triple of an arbitrary level set q is the sum of many mutually independent random variables. The tree nodes assigned to level set triple q have contributions from each

forest. We claim that the size of these contributions, $x_{j,q}$, are mutually independent over the forests. The value of $x_{j,q}$ is defined entirely by the level of the roots of the f_j and by the stretch counts chosen there, which are independent by definition. The level of the roots of f_j depends on the level of r_j and on the stretch count chosen there. Since the stretch counts are independent the level of the roots and thus the values of $x_{j,q}$ for any q are independent.

Now we can apply Lemma 10 by noting that each $x_{j,q}$ is less than $2^{\frac{2}{3}n}$ and that $E(x_{j,q}) = \frac{3M_j}{\log N}$. Let $X_q = \sum_{j=1}^{m} x_{j,q}$. Applying the lemma yields:

$$\begin{aligned} Pr(X_q \geq \beta \frac{N}{\log^2 N}) &\leq exp(-\frac{\beta N}{2^{\frac{2}{3}n}\log^2 N}) \\ &\leq exp(-\frac{\beta N^{\frac{1}{3}}}{\log^2 N}) \\ &\leq N^{-c'} \end{aligned}$$

A similar argument is valid for zone 0 and 2, so with probability greater than $1 - 3N^{-c'}$ the number of i-nodes coming from all zones to level set triple q is $O(\frac{N}{\log^2 N})$. Therefore with probability greater than $1 - 3nN^{-c'}$ every block in column i receives $O(\frac{N}{\log^2 N})$ nodes. And with probability greater than $1 - 3n^2N^{-c'} > 1 - N^{-c}$ every block in the tori receives $O(\frac{N}{\log^2 N})$ nodes. ■

Theorem 5 *With probability $1-N^{-c}$, $c > 0$, the above algorithm dynamically embeds every N-node tree T in the $\sqrt{P} \times \sqrt{P}$ grid so that each grid node receives $O(N/P)$ tree nodes, and such that the dilation is $O(\frac{\sqrt{P}}{\log N})$ and the congestion is $O(\frac{N}{\sqrt{P}\log N})$.*

Proof.

By Lemma 12 each block will receive $O(\frac{N}{\log^2 N})$ tree nodes with high probability. Since the choices of processor among one block are completely even, each processor receives $O(N/P)$ tree nodes with high probability.

The image of a tree edge can traverse at most 3 blocks in each dimension, therefore the dilation is bounded by $6\frac{\sqrt{P}}{\log N}$. The tree edges are mapped to the shortest path in the network with at most one turn. Denote the processor in row i and column j by $P_{i,j}$. Without loss of generality, consider a horizontal communication link ℓ between $P_{i,j}$ and $P_{i,j+1}$. From the dilation bound, at least one end point of every tree edge whose image traverses ℓ must lie within the interval $P_{i,j-3\sqrt{P}/\log N}$ through $P_{i,j+3\sqrt{P}/\log N}$. Since each grid node has $O(N/P)$ tree nodes and each tree node has at most degree 3, the total number of tree edges that can possibly go through ℓ is bounded by $O(\frac{N}{\sqrt{P}\log N})$. ■

The technique for two-dimensional grids can be easily extended to grids with multiple dimensions. In particular, for P-node grids with a fixed number k of dimensions, the bounds on dilation and congestion are $O(P^{1/k}/\log N)$ and $O(\frac{N}{P^{1-1/k}\log N})$ respectively.

Theorem 6 *With probability $1 - N^{-c}$, $c > 0$, the generalization of the above algorithm (which divides the grid into $\log N$ blocks along each dimension) dynamically embeds every N-node tree T in the $P^{1/k} \times \cdots \times P^{1/k}$ k-dimensional grid (constant k) so that each grid node receives $O(N/P)$ tree nodes, and such that the dilation is $O(P^{1/k}/\log N)$ and the congestion is $O(\frac{N}{P^{1-1/k}\log N})$.*

6 Randomized Lower Bound

We again consider the two–processor model for our randomized lower bound. Let $\mathcal{A}$ be any probabilistic on-line algorithm which is α–balanced, $1 \leq \alpha < 2$. For an N–node tree the algorithm guarantees that there are at most $\alpha\lceil\frac{N}{2}\rceil$ nodes in either processor. In what follows we show how to construct an N–node binary tree $\mathcal{T}_{\mathcal{A}}$ such that the expected number of cut edges created by $\mathcal{A}$ for $\mathcal{T}_{\mathcal{A}}$ is at least $(1-\alpha/2)^2 N/18\log N$. As a consequence, we can conclude that any balanced on-line tree embedding algorithm can be expected to to make $\Omega(N/\log N)$ cuts on the worst-case tree.

Theorem 7 *For every algorithm $\mathcal{A}$, and $\alpha > 1$, there exists a growth sequence ρ of length N such that if the on–line embedding of ρ is terminally α–balanced between two processors then the expected number of cut edges is $\Omega(N/\log N)$.*

Proof.

The worst-case tree consists of a sequence of complete binary trees that are grown as follows. We start with the depth–1 complete binary tree with 3 nodes. Let β_i be the random variable denoting the fraction of edges in the ith level that are cut edges and $r = \frac{1}{3}(1-\alpha/2)$. If $E(\beta_1) \leq r/\log N$, then we grow the tree one more level, to form the depth–2 complete binary tree with 7 nodes. On the other hand, if $E(\beta_1) > r/\log N$, then we start a new complete binary tree at the rightmost leaf of the current complete binary tree. See Figure 2 for an example.

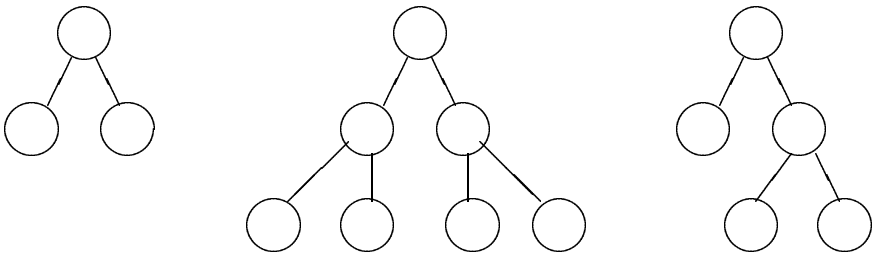

Figure 2: Tree growth examples

In general we continue growing $\mathcal{T}_{\mathcal{A}}$ using the same rule. If $E(\beta_i) \leq r/\log N$, then we grow $\mathcal{T}_{\mathcal{A}}$ by attaching two leaves to every level–i node.

If $E(\beta_i) > r/\log N$, then we grow $\mathcal{T}_{\mathcal{A}}$ by attaching two leaves to just the rightmost level–i node. In other words, we extend the current binary subtree by one level in the former case, and we start a new binary subtree in the latter case. The procedure stops when we have grown N nodes. Figure 3 illustrates one possible choice for a 32–node tree. Note that every level (except possibly the last) contains 2^a nodes, where $a \geq 0$.

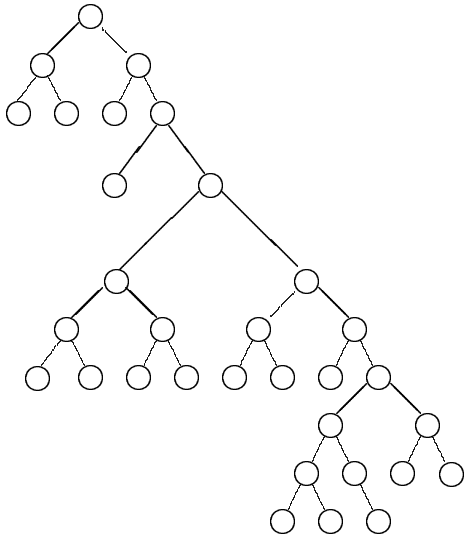

Figure 3: A 32-node example

For the purposes of our discussion, it is useful to consider $\mathcal{T}_{\mathcal{A}}$ as a collection of complete binary subtrees $\{T_i\}$. In particular, we define T_i to be the ith maximal complete binary subtree formed during the construction of $\mathcal{T}_{\mathcal{A}}$. We denote the last subtree formed as T_m. The last level of T_m may be partially empty. We also define n_i to be the number of nodes in T_i, less the rightmost leaf for $i < m$ since this node forms the root of the next tree.

Let C be the random variable denoting the number of cut edges created by $\mathcal{A}$ on $\mathcal{T}_{\mathcal{A}}$. In what follows we show that $E(C) \geq (1-\alpha/2)^2 N/18 \log N$ when $N = 2^n$. The proof is divided into two cases, depending on the value of $\sum_{i=1}^{m-1} n_i$.

Case 1. $\sum_{i=1}^{m-1} n_i \geq rN$ where $r = \frac{1}{3}(1-\alpha/2)$.

Let x_i be the random variable denoting the number of cut edges in the last level of T_i, $1 \leq i \leq m-1$. The last level of T_i has $\frac{1}{2}n_i$ edges and expected fraction of cut edges greater than $r/\log N$, therefore $E(x_i) \geq \frac{1}{2}n_i r/\log N$. Hence,

$$E(C) \quad \geq \quad \sum_{i=1}^{m-1} E(x_i)$$

$$
\begin{aligned}
&> \sum_{i=1}^{m-1} \frac{1}{2} n_i r / \log N \\
&= \frac{1}{2} r N (r / \log N) \\
&= (1-\alpha/2)^2 N / 18 \log N.
\end{aligned}
$$

Case 2. $\sum_{i=1}^{m-1} n_i < rN$.

Since $\sum_{i=1}^{m-1} n_i < (1-\alpha/2)N/3$, we know that $n_m > N - (1-\alpha/2)N/3 = (4+\alpha)N/6$, and the number of levels in T_m is $\log N - 1$. Let σ_j be the random variable denoting the fraction of cut edges at level j of T_m for $1 \leq j \leq \log N - 2$. By assumption, $E(\sigma_j) \leq r/logN$. Also, let s_j be the random variable denoting the number of nodes on level j that remain connected to the root of T_m if all the cut edges are cut. Then, $s_0 = 1$ and $s_j \geq 2s_{j-1} - \sigma_j 2^j$ for $1 \leq j \leq \log N - 2$. Solving the recurrence for s_j, we find that $s_j \geq 2^j(1 - \sum_{i=1}^{j} \sigma_i)$ for $1 \leq j \leq \log N - 2$.

Since $r < 1/6$, T_m is a complete binary tree with at most $1/6N$ nodes not in the last level. Therefore $s_{\log N-1} \geq 2s_{\log N-2} - Y - rN$, where Y is the random variable denoting the number of cut edges on the last level of T_m. Let R be the random variable denoting the number of nodes in T_m that are still connected to the root of T_m when all the cut edges are cut.

$$
\begin{aligned}
R &\geq \sum_{j=0}^{\log N-1} s_j \\
&\geq (\sum_{j=0}^{\log N-2} 2^j (1 - \sum_{i=1}^{\log N-2} \sigma_i)) + s_{\log N-1} \\
&\geq (\sum_{j=0}^{\log N-2} 2^j (1 - \sum_{i=1}^{\log N-2} \sigma_i)) + 2s_{\log N-2} - Y - rN \\
&\geq (\sum_{j=0}^{\log N-1} 2^j (1 - \sum_{i=1}^{\log N-2} \sigma_i)) - Y - rN \\
&= (N-1)(1 - \sum_{i=1}^{\log N-2} \sigma_i) - Y - rN
\end{aligned}
$$

Since the largest component of $\mathcal{T}_{\mathcal{A}}$ can have size at most $\alpha N/2$ once the cut edges have been removed, we know that $R \leq \alpha N/2$. Thus we have that

$$
Y \geq (N-1)(1 - \sum_{j=1}^{\log N-2} \sigma_j) - rN - \alpha N/2
$$

By definition, $C \geq Y$, and, therefore

$$
E(C) \geq E(Y)
$$

$$
\begin{aligned}
&\geq (N-1)(1-(\log N-2)r/\log N) - rN - \alpha N/2 \\
&= (1-r+2r/\log N)(N-1) - rN - \alpha N/2 \\
&\geq (1-r-r-\alpha/2)N - 1 \\
&= 1/3(1-\alpha/2)N - 1.
\end{aligned}
$$

This concludes the proof that $\mathcal{A}$ is expected to create at least $\frac{(1-\alpha/2)^2 N}{18 \log N}$ cut edges when embedding $\mathcal{T}_{\mathcal{A}}$ on-line among two processors. ■

The preceding result can be generalized to show that for any on-line randomized algorithm for partitioning an N–node binary tree into components of size at most $O(N/P)$ the expected number of cuts is at least $\Omega(N/\log \frac{N}{P})$ in the worst–case.

Theorem 8 *For every algorithm $\mathcal{A}$, and $\alpha > 1$, there exists a growth sequence ρ of length N such that if the on–line embedding of ρ is terminally α–balanced among P processors then the expected number of cut edges is $\Omega(N/\log \frac{N}{P})$.*

Proof.

The proof is nearly identical to that above, except that we use a threshold of $\Theta(1/\log \frac{N}{P})$ instead of $\Theta(1/\log N)$ when deciding whether or not to start a new complete binary tree at each level of $\mathcal{T}_{\mathcal{A}}$. From Theorem 4 this bound is tight, up to constant factors, for all P, $2 \leq P < 2N$. ■

7 Conclusion

The execution of divide-and-conquer type algorithms on multicomputers requires a simple strategy for distributing the subprocesses as they are created. Ideally the distribution would give a balanced load and not require large communication overhead. We have shown that on grid and butterfly networks the worst-case dilation of any deterministic balanced algorithm is as large as the diameter of these networks.

Allowing randomization yields some improvement. Both the dilation and the congestion can be improved by a factor of $\log N$. As shown in [4] this reduces the dilation for the butterfly to a constant. We have also shown that the dilation and expected worst-case congestion for meshes cannot be improved by more than a logarithmic factor.

These large overheads for grids leads to the question of whether there are algorithms with better performance for the kinds of trees that arise in practice.

Acknowledgments

The authors thank Jin-Yi Cai for helpful discussions. A portion of the work was done while Sandeep Bhatt was visiting Caltech, and supported there by DARPA Order Number 6202, monitored by ONR under contract N00014-87-K-0745. Sandeep Bhatt, David Greenberg and Pangfeng

Liu, then at Yale University, were supported in part by Air Force grant AFOSR-89-0382, NSF grant CCR-88-07426, and NSF/DARPA grant CCR-89-08285. Tom Leighton was supported at MIT by Air Force grant AFOSR-89-0271, Army contract DAAL-03-86-K-0171, and DARPA contracts N00014-89-J-1988 and N00014-87-K-825.

References

[1] S. Bhatt and J. Cai, *Taking random walks to grow trees in hypercubes*, Journal of the ACM, 40 (1993), pp. 741–764.

[2] S. Bhatt and C. Leiserson, *How to assemble tree machines*, in Advances in Computing Research 2, 1984, pp. 95–114.

[3] R. Karp and Y. Zhang, *A randomized parallel branch-and-bound procedure*, in 20th Annual ACM Symposium on Theory of Computing, 1988.

[4] T. Leighton, M. Newman, A. Ranade, and E. Schwabe, *Dynamic tree embedding in butterflies and hypercubes*, SIAM Journal on Computing, 21 (1992), pp. 639–654.

[5] A. Ranade, *A simpler analysis of the Karp-Zhang parallel branch-and-bound method*, Tech. Rep. UCB/CSD 90/586, University of California, 1990.

[6] ——, *Optimal speedup for backtrack search on a butterfly network*, in 3rd Annual ACM Symposium on Parallel Algorithms and Architecture, 1991.

[7] A. Rosenberg and L. Snyder, *Bounds on the costs of data encodings*, Mathematical Sytems Theory, 12 (1978), pp. 9–39.

[8] C. Seitz, *Multicomputers*, Addison–Wesley, 1990, pp. 131–201. In *Developments in Concurrency and Communication*, C.A.R Hoare (ed,).

[9] I. Wu and H. Kung, *Communication complexity for parallel divide and conquer*, in IEEE Symposium on Foundations of Computer Science, 1991, pp. 151–162.

[10] Y. Zhang, *Parallel Algorithms for Combinatorial Search Problems*, PhD thesis, U.C. Berkeley, 1989.