Tight Bounds for On–line Tree Embeddings

(Extended Abstract)

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

[1] Department of Computer Science, Yale Univ., New Haven CT 06520.

[2] Computer Science Department 256-80 Caltech, Pasadena CA 91125.

[3] Mathematics Department and Laboratory for Computer Science, MIT, Cambridge MA 02139

Abstract

Many tree–structured computations are inherently parallel. As leaf processes are recursively spawned they can be assigned to independent processors in a multicomputer network. To maintain load balance, an on–line mapping algorithm must 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 performance (load–balance) 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, although we find that the performance of randomized on-line algorithms to be somewhat better than that of deterministic algorithms. 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 or the evaluation of functional expressions and game–trees. As the computation evolves, the corresponding tree grows and shrinks. Each node of the tree can recursively spawn subprocesses, and possibly also communicate with its parent. The most significant feature of tree computations is that each node can spawn children independently 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 when many processes are spawned simultaneously and each travels over a long distance in the network to its assigned processor. For any given network, one might reasonably suspect a trade–off between performance (load distribution) and efficiency (congestion).

For the boolean hypercube and its derivative networks, the butterfly and cube-connected-cycle networks for example, simple and efficient algorithms have recently been developed [1, 3]. 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 $O(1)$ [3].

In this paper we extend our study of on–line embeddings to different multicomputer 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 [3]. We also adapt the techniques of [3] 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. The existence of small off-line bisectors for trees suggests that with just two processors only a small amount of communication should be needed. However, we show that even in this simple case large communication overhead is inevitable for on-line embeddings.

The remainder of this abstract 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 presents randomized algorithms with improved performance. Section 5 examines lower bounds for randomized algorithms.

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 multicomputer 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.

The on–line embedding algorithm is 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 multicomputers [4]. Disallowing process migration is a common policy in practice; migration can conceivably lead to thrashing and cripple a large multicomputer network.

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 ith 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 2.

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.

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 length path 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 the size P of the network.

An on–line embedding for a growth sequence ρ is *continuously α–balanced*, if for every i, the ith 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.

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

3 Deterministic Lower Bounds

3.1 The two processor case

Given a deterministic on-line algorithm we show how to construct an N-node tree for which the algorithm either requires $\Theta(N)$ cut edges or fails to achieve terminal balance. 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, 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 in 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. This ensures that the construction outlined above is well-defined.

The claim that the number of cut edges grows

linearly with the size of tree, under the requirement of terminal balance, 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 the ith layer if the path from the root to the node contains $i-1$ cut edges. The root is defined to be in layer 1. Let n_i be the number of nodes in the ith layer. Each node spawned during phase i lies either in layer i or in layer $i+1$.

LEMMA 3.1. *For every deterministic algorithm $\mathcal{A}$, and $\alpha < 2$, 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 the number of cut edges is $\Omega(N)$.*

Proof. Let $\alpha = 2-\delta$ and choose $\epsilon > 0$ so that $\epsilon < \delta/4$. Using the adversarial strategy above, grow a tree with N nodes, $N > \alpha/(2-4\epsilon-\alpha)$.

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

In the first 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 $N - \alpha\lceil N/2\rceil > \epsilon N$ white nodes, and consequently at least ϵN cut edges.

In the second case let n_i be the number of nodes in the ith layer. We again distinguish between two subcases: either $\sum_1^{k-1} n_i \geq \epsilon N$, or $\sum_1^{k-1} n_i < \epsilon N$.

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 the first subcase the number of cut edges is greater than ϵN.

In the second 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 $n_{k+1} > N(1-\epsilon) - \alpha\lceil N/2\rceil > \epsilon N$.

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 3.1 implies that every deterministic on-line, load-balancing algorithm requires a linear number of cut edges on some finite sequence. Since the theorem holds for terminal balance, it holds for continuous balance as well.

The on–line lower bound contrasts sharply with the off–line bound. Every length–N growth sequence can be terminally $\frac{4}{3}$–balanced with just one cut. Less trivially, we can show that every length–N growth sequence can be *continuously* α–balanced with $O(\log^2 N)$ cuts, for any $1 < \alpha < 2$. This is achieved using multi–color bisectors [2].

Lemma 3.1 can be also used to construct one infinite sequence such that every continuously–balanced algorithm is forced, on infinitely many prefixes, to make linearly many cuts. This infinite sequence is constructed using a standard diagonalization argument.

Multiple processors

Lemma 3.1 can be extended to a bounded number of processors. With P colors instead of two, we use a similar argument to the one above. The only difference is that each phase grows only one subtree, rooted at a leaf grown in the previous phase. A phase ends when all the leaves grown in that phase are colored differently from the root.

LEMMA 3.2. *For every deterministic algorithm $\mathcal{A}$, $P \geq 2$, and $\alpha < P$, 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 among P processors then the number of cut edges is $\Omega(N)$.*

Remark. In some computations it is important to evenly distribute only the leaves. Lemma 3.2 can be extended to this case as well.

3.2 Congestion and Dilation Bounds

We use Lemma 3.2 to derive lower bounds on the congestion and dilation of balanced algorithms as follows. We assume that the tree size, N, is $\Omega(P)$ where P is the size of the network. We partition the given network into B blocks of equal size. We refer to an edge that connects nodes in different blocks as a *cross-link*.

An α–balanced embedding of an N node tree assigns at most $\alpha\lceil N/B\rceil$ tree nodes per block.[1] By Lemma 3.2, 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.

The P–node two-dimensional grid can be partitioned into B blocks with $B\sqrt{P}$ cross links. Consequently, the congestion is $\Omega(N/\sqrt{P})$. For the P–node butterfly $C = O(P/\log P)$; the congestion is $\Omega(\frac{N\log P}{P})$.

It is equally easy to obtain lower bounds on dilation.

[1] Given α we choose a suitable value for B so that each node of the network can receive at most a small fraction of the tree nodes.

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. Now 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, $d = \Omega(\log P)$.

THEOREM 3.1. *For every deterministic algorithm $\mathcal{A}$, and $\alpha > 1$, 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 $\Omega(\frac{N\log P}{P})$.*

The bounds on dilation are tight since they match the diameters of the respective networks. To achieve the bounds on congestion a deterministic algorithm uses a single global fetch-and-add register to distribute the tree nodes to the mesh or butterfly in some fixed order. The global register is implemented via a spanning tree and contributes at most constant congestion. Further details will be in the final paper.

4 Randomized Algorithms

4.1 n-way Balancing The key tool for our randomized algorithms is the *n–way balancing transformation* [3] 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 $i \equiv 0 \ mod\ n/3$. For each distinguished node v we pick a random number $S(v)$ between 0 and $n/3-1$ inclusively 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 $p_0, p_1 \ldots p_{n-1}$ in clockwise order. The algorithm embeds the tree nodes in level set i of $B(T)$ into processor p_i.

This transformation embeds every sufficiently large binary tree into a ring evenly with high probability, and with dilation two.

LEMMA 4.1. ([3]) *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 $O(N/\log N)$ nodes of a binary tree, 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 nodes. We use $\log N$-way balancing to assign tree nodes to groups, and a deterministic strategy is used within each group. For the length–L ring, the dilation is bounded by $O(L/\log N)$, while the congestion is $O(N/\log N)$. The bound on dilation is optimal; any mapping of the N–node complete binary tree on the L–node ring requires dilation $\Omega(L/\log N)$. 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 5, on the expected value of the congestion for any randomized algorithms.

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 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 each node in every $\frac{1}{3}\log\frac{N}{P}$–th level set of the transformed tree is mapped to a processor chosen uniformly at random among the P processors. Those nodes not in these level sets are embedded into the processor where their parents are embedded. Since only edges at every $\frac{1}{3}\log\frac{N}{P}$–th level are cut, the total number of cut edges is $O(N/\log\frac{N}{P})$. Although we omit details in this abstract, we can show that for $P = O(N/\log N)$, the tree nodes are equitably distributed among the P processors. For

larger values of P, the same bound is achievable, but the algorithm becomes more complicated and requires global information.

4.2 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 $\Omega(\sqrt{P}/\log N)$. 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 on-line deterministic algorithm 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 5 that the expected value of the congestion is $\Omega(\frac{N}{\sqrt{P}\log N})$. The randomized algorithm presented here will meet this bound.

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 the 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 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.

THEOREM 4.1. *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 Sketch. Using Lemma 4.1 we first argue that each column of blocks receives a total of $O(\frac{N}{\log N})$ tree nodes with high probability. Next, given that each column has $O(\frac{N}{\log N})$ nodes, we bound the probability that the distribution in some column is uneven. In particular, we show that, with probability $1 - N^{-c}$, $c > 0$, each block receives $O(N/\log^2 N)$ tree nodes. With deterministic allocation within each block, each grid node receives $O(N/P)$ tree nodes. (As in the deterministic algorithm sketched at the end of Section 3 the allocation within blocks will require the simulation of a global register for each block.)

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, pick 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.

5 Randomized Lower Bound

We again consider the two–processor model for our 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 $\alpha N/2$ 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 5.1. *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 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 α_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 $E(\alpha_1) \leq r/\log N$, then we grow the tree one more level, to form the depth–2 complete binary tree with 7 nodes. On the other hand, if $E(\alpha_1) > r/\log N$, then we start a new complete binary tree at the rightmost leaf of the current complete binary tree. See figure 1 for an example.

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

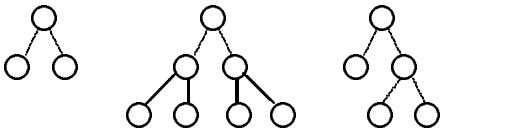

Figure 1: Tree growth examples

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 2 illustrates one possible choice for a 32–node tree. Note that every level (except possibly the last) contains 2^a nodes, where $a \geq 0$.

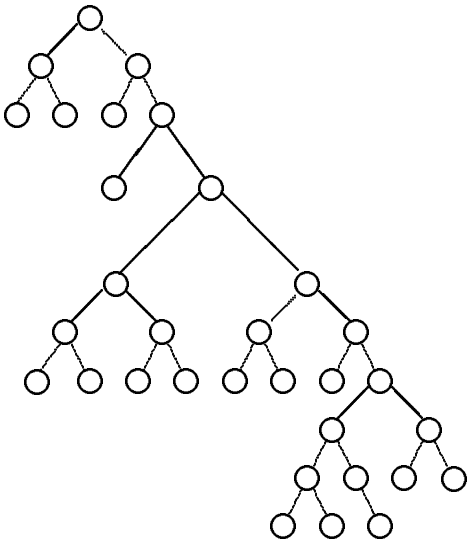

Figure 2: A 32-node example

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,

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

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

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

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

Remark. The preceding result can be generalized to show that any on-line algorithm for partitioning an N–node binary tree into components of size at most N/P is expected to create at least $\Omega(N/\log \frac{N}{P})$ cut edges for the worst–case tree. The proof is nearly identical to that given 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 $\mathcal{T}_{\mathcal{A}}$. In addition, we only need to worry about the top $\log \frac{N}{P}$ levels of T_m in the analysis for case 2. As we showed earlier in the paper, this bound is tight, up to constant factors, for all P, $2 \leq P < N/2$.

6 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. In the final version we will show that the same is true for the shuffle-exchange network as well.

Allowing randomization yields some improvement. Both the dilation and the congestion can be improved by a factor of $\log N$. As shown in [3] 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.

Acknowledgements

The authors thank Jin-Yi Cai for helpful discussions. Sandeep Bhatt was supported at Caltech by DARPA Order Number 6202, monitored by ONR under contract N00014-87-K-0745. Sandeep Bhatt, David Greenberg and Pangfeng Liu were supported at Yale 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] Sandeep N. Bhatt and Jin-Yi Cai. Take a walk, grow a tree. In *29th Annual IEEE Symposium on Foundations of Computer Science*, pages 469–478, 1988.

[2] Sandeep N. Bhatt and Charles E. Leiserson. How to assemble tree machines. In *Advances in Computing Research 2*, pages 95–114, 1984.

[3] Tom Leighton, Mark Newman, Abhiram G. Ranade, and Eric Schwabe. Dynamic tree embedding in butterflies and hypercubes. In *1989 ACM Symposium on Parallel Algorithms and Architectures*, pages 224–234, 1989.

[4] Charles L. Seitz. Multicomputers. To appear in *Developments in Concurrency and Communication*, C.A.R Hoare (ed) Addison–Wesley, 1990. pp 131-201.