An Approach to Emulating Separable Graphs

Bojana Obrenić

Department of Computer Science
University of Massachusetts at Amherst

September 17, 1991

Acknowledgment of Support: This research was supported in part by NSF Grants CCR-88-12567 and CCR-90-13184.

Author's Address: Department of Computer Science, University of Massachusetts, Amherst, MA 01003; obrenic@cs.umass.edu

Remark: A preliminary version of this paper was presented at the *3rd ACM Symp. on Parallel Algorithms and Architectures*, July 22-24, 1991, in Hilton Head, S.C.

Abstract

We present an *embedding technique* for bounded-degree guest graphs that have *sublinear separators* (multicolor recursive bisectors). We introduce an intermediate structure called a *cell tree*, and we employ it as a generic host for bounded-degree separable graphs. In these embeddings, the node assignment is derived from the separator-based decomposition of guest graphs; the edge routing is achieved through the coordination of *global* and *local* phases. Together with the embeddings, we construct communication schedules with *queue-size* 1 (no dynamic congestion).

We instantiate our technique by embedding the generic hosts, cell trees, into specific host graphs. With *shuffle-like* hypercube-derivative networks as hosts we obtain new embeddings of our guest graphs with constant expansion, having both dilation and congestion logarithmic in the size of the multicolor bisector. The *congestion* of these embeddings is *exponentially smaller* than previously known, thereby exponentially speeding up the emulations of bounded-degree separable graphs by shuffle-like networks. When applied to *hypercubes* as hosts our technique also produces new embeddings, whose expansion constants are better than those known so far.

1 Introduction

Motivation. The goal of this study is to understand and compare the computational powers of processor arrays based on various interconnection topologies. We compare the powers of processor arrays by determining how efficiently each of them can *emulate* the others. To formalize our notion of emulation, we represent processor arrays by (undirected) graphs, whose nodes correspond to processors and whose edges correspond to interprocessor communication links. Emulations among processor arrays are then derived from embeddings among their underlying graphs (cf. Section 1.1). An embedding maps nodes and edges of the emulated array (guest) into nodes and paths of the emulating array (host), thereby associating host processors to guest processors, and paths in the host network to links in the guest network. The slowdown of emulations is measured by embedding costs (cf. Section 1.1)—the slowdown grows with length of these paths (dilation) and with number of paths contending for a single host link (congestion). Emulations provide automatic, step-by-step translations of guest programs into functionally equivalent host programs; the only cost thus incurred is the emulation slowdown. One way to make programs more independent of the specifics of interconnection networks is, therefore, to construct efficient emulations among broad classes of networks. In this paper, we show how a wide class of *arbitrary* networks can be efficiently emulated by a *generic* host; the generic host then can be efficiently emulated by a variety of networks, as we demonstrate on two network families.

Previous results. The first efficient emulations of arbitrary bounded-degree networks with sublinear node separators (multicolor recursive bisectors, cf. Section 2.1) appeared in [4], where it was proved that such graphs can be embedded into butterflies with dilation logarithmic in the size of their multicolor recursive bisectors and with constant expansion. These embeddings essentially relied on the embedding of complete binary trees into butterflies (with dilation 6 and expansion 8); the embeddings in [4] were actually a composition of this embedding and embeddings of guests graphs into complete binary trees. However, the emulations of [4] incurred slowdown as great as *linear* in the size of the multicolor bisector, because the *congestion* of these embeddings was of that order (cf. Sections 1.1, 2.1). This deficiency was subsequently remedied, by proving that *every* embedding of a bounded-degree graph in a butterfly which has good dilation (cf. Section 2.1) can be transformed into an embedding with simultaneous good dilation *and* good congestion (details of this proof are available in [3]). Since emulation slowdown is proportional to the sum of dilation and congestion ([13], cf. Section 1.1), these results provided emulations of bounded-degree graphs by butterflies with slowdown logarithmic in the size of their multicolor bisectors.

The upper bounds for emulations by butterfly networks of [3, 4] hold unchanged for hypercubes, since hypercubes can emulate butterflies with only negligible slowdown [10]. However, the techniques of [4] are not applicable to arbitrary host networks. In particular, when it comes to emulations by *shuffle-like networks* (cf. Section 4), the approach of [4] is not as useful as it is for butterfly-like hosts: Although it is known

from [4, 3] how to embed our guest graphs into complete binary trees, and although shuffle-like graphs essentially contain like-sized complete binary trees, *no way has been found to control congestion* of embeddings in shuffle-like graphs. These graphs have thus been able to emulate bounded-degree separable graphs exponentially slower than their butterfly relatives.

Our results. In this paper, we present a new technique for emulating bounded-degree separable graphs. To this end we devise a notion of *cell trees* (cf. Section 2) which play the part of generic hosts for these guests. It should be stressed that we do not propose cell trees as new interconnection topologies; rather, cell trees abstract the ability of hosts to emulate efficiently bounded-degree separable graphs; indeed cell trees are host graphs *organized* to emulate these guests. Our organization is characterized by three parameters (cell slowdown, transfer fraction, and transfer slowdown, cf. Section 2.2). On the one hand, we show how an arbitrary bounded-degree separable graph can be emulated by a cell tree; we express the emulation's slowdown in terms of these three parameters. On the other hand, we show how to compute these parameters for cell trees constructed in specific hosts; we compute them for hypercubes (Section 3) and shuffle-like graphs (Section 4). The upper bounds that we obtain for shuffle-like graphs match those known for butterflies [3, 4] and provide the desired optimal speedup. The embeddings in hypercubes that we obtain are also new and differ from those previously known by having improved expansion constants (cf. Section 1.1).

Advantages of our approach. We have divided the task of embedding bounded-degree separable graphs into two subtasks: The first subtask is the canonical representation of guest graphs—the bucket trees—which we construct by powerful graph decomposition mechanisms (Section 2.1). The second subtask is organization of host graphs, which is directed by the *intermediate structure* of the cell tree (Section 2.2). We find efficient embeddings of guest graphs into cell trees, thereby reducing the original problem to an easier one: embedding the cell tree into the host graph.

There are two major reasons that cell trees are efficient hosts for bounded-degree separable graphs. First, cell trees fully exploit the information contained in the *bucket trees*. More specifically, the bucket tree obtained via recursive decomposition of the guest graph already determines the dilation of the embedding; the assignment of guest nodes to cell-tree nodes derives directly from the bucket tree and honors these dilation constraints. Second, routing within the cell tree obeys a coordinated regimen which alternates *global and local routing* phases, thereby avoiding congestion.

We claim that the remaining problem of embedding cell trees into specific hosts is easier than the original problem of embedding arbitrary bounded-degree separable graphs into the hosts. To wit, this embedding problem further reduces to efficient *deterministic routing of a small class of specific permutations*. We exhibit efficient routings of these permutations for two sample host networks, thus producing cell trees with good parameters for these hosts and enabling emulations much faster than previously known.

In our pursuit of topology-independent treatment of parallel architectures we recog-

nize bucket trees and cell trees as useful representations of guest and host graphs. These representations are succint and simple as they consist of few numerical parameters; at the same time they are rich enough to abstract the relevant graph properties.

For a given guest graph, the task of constructing a bucket tree (and of deriving its matching cell tree) are both independent of the host. The task of embedding the cell tree into a host is independent of the guest. The role of the cell tree is analogous to that of the tree of meshes in [12, 8]: The efficient layout for the tree of meshes parallels the construction of an efficient cell tree; reducing the problem of laying out an arbitrary (bounded-degree separable) graph to the problem of laying out the tree of meshes parallels our reducing the problem of emulating an arbitrary (bounded-degree separable) graph to the problem of emulating the cell tree.

We review our study's computation model in the following subsection. Section 2 presents the general technique; Sections 3 and 4 apply it to specific networks.

Conventions. We denote by $|G|$ the number of nodes of graph G. We let $\lg x =_{\text{def}} \lfloor \log_2 x \rfloor$. $Z_2 =_{\text{def}} \{0, 1\}$; for integer $k \geq 0$, we denote by Z_2^k the set of all strings of length k over Z_2. The *complete binary tree* $T(h)$ of *height* h has node-set $\bigcup_{0 \leq k \leq h} Z_2^k$ and edges connecting each $y \in Z_2^k$, $0 \leq k < h$, to its *children* $y0$ and $y1$. The *root* of the tree $T(h)$ is the empty string λ; the *leaves* of $T(h)$ are all nodes $y \in Z_2^h$. The 2^k nodes $x \in Z_2^k$, for $0 \leq k \leq h$, reside at *level* k.

1.1 Computation Model, Emulations, and Embeddings

Processor Arrays. We study *arrays* of *identical processing elements*, each associated with a *local memory* module, connected by an *interconnection network*. The computation of such a processor array develops as a sequence of *pulses*; each pulse is either a *computation* or a *communication* step. Computation steps involve only local memories of processing elements; during communication steps each processing element may send a *message* to its neighbor(s). As is customary, we represent processor arrays as undirected *graphs*, whose *nodes* stand for processing elements, and whose *edges* stand for *interprocessor communication links*. It is possible to (consistently) extend our interpretation so as to allow us to view the graphs we study as dependency graphs of parallel computations; this view gains significance when our results hold for arbitrary guest graphs.

We investigate and compare the powers of processor arrays by determining how efficiently each array can *emulate* the other. To explain our emulation setup, we recall its core formal notion: *embedding* one graph in another [14].

Embeddings. An embedding of *guest* graph G in *host* graph H comprises two mappings: an injective *assignment* of the *nodes* of G to *nodes* of H, plus a *routing* that assigns to each *edge* of G a *path* in H connecting the images of the edge's endpoints. Efficiency of an embedding is described in terms of its:

- *dilation* — the length of the longest path in H that routes an edge of G,

- *congestion* — the maximum number of edges of G routed over any single edge of H,
- *expansion* — the ratio $(|H|/|G|)$.

Emulations. In the course of an emulation, each host processor in H emulates the guest processor which is its preimage in G; this assignment between processors is fixed throughout the emulation. For every step of a processor in G, its image in H executes a sequence of one or more steps. At any time, all processors of the host array emulate the same step of the guest array; the emulation of the next guest step may begin only after all the host sequences emulating the current guest step have ended. This correspondence between steps in the guest computation and sequences of steps in the host computation is also fixed: we neither allow host processors to perform any other computation nor do we require any component of the state of any guest processor at any time to be made available (unless mandated by the guest computation itself) to any host processor other than one assigned to it.

We assume that processors of G and H have equal power; computation steps, therefore, incur no slowdown, as one host computation step is sufficient to emulate one guest computation step. Guest communication steps are emulated by sequences of host communication steps, that take messages for the guest over the paths in H that route edges of G. The communication *schedule* specifies the host communication sequences by naming the links of H that are crossed at every step of these sequences. The *dynamic congestion* of the schedule is the maximum number of messages that are simultaneously waiting to be transmitted by any node of H; the *queue-size* of the schedule measures the space overhead on saving these messages. The emulation *slowdown* is determined by the length of the longest communication sequence. When there is an embedding of graph G into graph H with dilation Δ and congestion K, then it is straightforward to construct a schedule with slowdown $O(\Delta K)$. In [13] it is shown (by a nonconstructive proof) that communication steps can be orchestrated to produce a schedule with slowdown $O(\Delta+K)$. We provide explicit constructions of our communication schedules with slowdown of the latter order and without dynamic congestion (or additional queues).

We stress that our sole concern is the cost (in time and space) of overcoming structural mismatches among the underlying graphs of processor arrays. To isolate this cost, we insist that the processing power of guest and host processors must be identical. The association between the emulated processor and its image in the emulating array thus implies that representations of states of the two processors match exactly in structure and size, modulo overhead on supporting the emulation, suffered by the host processor. In particular, their (potentially unbounded) memories are in exact correspondence; the same holds for their I/O streams. This uniform assignment, together with our emulation algorithm, affords us uniform and automatic translation of guest program steps into host program steps; the translation is independent of the meaning of the program and therefore valid in general. If the assumption of equal power is relaxed, by allowing host

processors to be more powerful than those of the guest, one can define alternative notions of emulation (cf. [11, 16]) that require more sophisticated computations by the host processors and more complicated processor assignments (one-to-many). In this relaxed environment there may be an efficient emulation of the guest processor array by the host processor array even when there is no efficient embedding of the guest graph into the host graph. The price of such relaxation is that it forces semantic restrictions on the guest programs that can be translated for the host; e.g., the emulations of [11] are not efficient when processor local memories are unbounded (cf. [16]), nor when processors are allowed to perform on-line I/O.

Our emulation requires that each *host* processor at any communication step can send a message to only *one* of its neighbors. Still, our upper bounds on communication slowdown hold for the case where each *guest* processor in a single step may send a message to *all* of its neighbors. This conservative assumption obviates the variations within the standard model as to the number of communication links that may be active at any step.

2 Emulations by Cell Trees

We commence by elaborating on two structures central to our approach: *guest-oriented* bucket-trees and *host-oriented* cell-trees.

2.1 Preliminaries on Decomposing Guest Graphs

We use two decomposition mechanisms to achieve the desired representation of our guest graphs: the recursive decomposition into like-sized subgraphs based on node-separation, and the decomposition into matchings via edge-coloring. We start by reviewing the more subtle of these mechanisms, the recursive bisection.

Remark. Although we phrase our argument in terms of graph *separators*, it is the balanced decomposition that is essential, not the instrument used to effect it. This means that nothing precludes substitution of *bifurcators* [8] for separators, when convenient.

Definition 1 *Let α be a real such that $0 < \alpha \leq 1/2$, and let S be a nondecreasing integer function. The graph G has an* α-**separator** *function S either if $|G| \leq 1$ or there is a set of no more than $S(|G|)$ nodes whose removal partitions G into subgraphs, each having no more than $(1-\alpha)|G|$ nodes and the separator function S. If G has the separator function S, then we say that $S(|G|)$ is the* **size** *of the separator of G.*

Definition 2 *Let k be a positive integer and let R be a nondecreasing integer function. The graph G has a* k-**color recursive bisector** *function R either if $|G| \leq 1$, or for every k-coloring of the nodes of G there exists a set of no more than $R(|G|)$ nodes whose removal partitions G into subgraphs G_1 and G_2, each having no more than $\lceil(|G|/2)\rceil$*

nodes and the recursive bisector function R, while $|(|G_1^\ell| - |G_2^\ell|)| \leq 1$, where $1 \leq \ell \leq k$ and $|G_i^\ell|$ is the number of nodes of the subgraph G_i colored by color ℓ. If G has the k-color recursive bisector function R, then we say that $R(|G|)$ is the **size** *of the k-color recursive bisector of G.*

All graphs having a separator function have also a k-color recursive bisector function of similar order. The proof of this fact employs the well known techniques of [8]; we just state the fact as the following.

Proposition 1 *For any integer k, any graph G that has a $(1/3)$-separator of size $S(|G|)$ has a k-color recursive bisector of size $R(|G|) = O(k \sum_i S(|G|/2^i))$. Therefore, $R(|G|) = O(kS(|G|)\lg|G|)$; when $S(|G|) = |G|^{\Omega(1)}$, then $R(|G|) = O(kS(|G|))$.*

The bucket tree originates in [5]; it has been successfully used in [4, 6, 7].

Definition 3 *Let graph G have maximum degree d and a $(d+1)$-color recursive bisector function R. The* **bucket tree** *$B^{(b)}$ for G is a complete binary tree of height $\lg|G|$, each of whose level-ℓ nodes has* **bucket capacity**

$$V(\ell) = bR(|G|/2^\ell)$$

for some constant b. Nodes of $B^{(b)}$ are called **buckets**.

The bucket capacity is meant to be interpreted as the number of nodes of G assigned to the bucket by some mapping. We denote by $B_x^{(b)}$ the nodes of G thus mapped to bucket x. Where the value of the constant b is implied by the context, we write B for $B^{(b)}$. The role of the bucket tree becomes clear from the following fact [4].

Proposition 2 *Let graph G of maximum degree d have a $(d+1)$-color recursive bisector function R. There exists a bucket tree $B^{(b)}$ for G such that the nodes of G can be mapped to the buckets of the bucket tree $B^{(b)}$ while:*

(a) *the number of nodes mapped to each bucket equals the bucket capacity;*

(b) *nodes that are adjacent in G are mapped to buckets that are at most distance d apart in the bucket tree, one of them being an ancestor of the other.*

The upper bounds in [4] on the cost of embedding bounded-degree separable graphs into butterflies are obtained via a two-step algorithm. In the first step, the bucket tree B of graph G is embedded into the complete binary tree T of height $\lg|G| + 1$, with dilation $O(\lg R(|G|))$, expansion 2, and congestion $O(dR(|G|))$. In the second step, T is embedded into the butterfly with dilation 6, expansion 8, and congestion 1. The obstructive congestion inherited from embedding B into T is removed *within* the butterfly by means of the following property of butterflies [3]:

Given a dilation-Δ embedding of a graph G with maximum degree d into the butterfly, there is an edge routing (for the same node assignment) with simultaneous dilation $O(\Delta)$ and congestion $O(d\Delta)$.

No equivalent to this property of reducing congestion via rerouting is known for shuffle-like graphs. Therefore, even though T can be embedded into shuffle-like graphs at virtually no cost, the congestion inherited from embedding B into T precludes emulations as efficient as those obtained with butterflies as hosts. At this point we give up the complete binary tree for the cell tree.

2.2 Cell Trees in Host Graphs

Cell trees capture those emulation capabilities of hosts that we deem essential for emulating separable graphs. We proceed by defining cell trees.

Definition 4 *Let H be a graph, c and h integer constants, and C a subset of nodes of H. Partition C into $2^{h+1}-1$ equal-size parts, called* **cells**, *each of cardinality c, and label the parts by distinct names from the set of nodes of the complete binary tree $T(h)$. The triplet (C,h,c) is the* **cell tree** *of* **height** *h and* **cell capacity** *c in H.*

Where no ambiguity may arise, we refer to the cell tree (C,h,c) by name of its node set C; a cell labeled by node x of the complete binary tree $T(h)$ is denoted by C_x.

The following two definitions put forward essential parameters of cell trees; these are the parameters that determine how successful hosts are while emulating separable guests.

Definition 5 *For graph H with cell tree (C,h,c), let the set of* **cell permutations** *$\mathcal{P}$ consist of the permutations of nodes of H that fix cells setwise. The* **cell slowdown** *of cell tree (C,h,c) in H is the maximum slowdown required to route any permutation in $\mathcal{P}$.*

Definition 6 *For graph H with cell tree (C,h,c), and a real $f \leq 1/2$, let $\mathcal{T}(f)$ be the set of all permutations of nodes of H that map at least (fc) nodes of each cell (other than the root cell) into nodes of its parent cell. The* **f-transfer slowdown** *of cell tree (C,h,c) in H is the minimum slowdown required to route some permutation in $\mathcal{T}(f)$. Call the permutation for which this minimum occurs the* **f-transfer permutation**, *call f the* **transfer fraction**, *and call the (fc) nodes in each cell mapped by the f-transfer permutation to the parent cell the* **f-transfer nodes**.

The rest of this Section is devoted to establishing our general result on the cost of emulating separable graphs of bounded degree. We express the emulation slowdown solely in terms of cell tree parameters—the cell slowdown and the f-transfer slowdown, for some transfer fraction f. Translating this cost into measures that refer to specific hosts requires constructing efficient cell trees in these hosts and computing cell tree parameters. Sections 3 and 4 present examples of such translation.

Theorem 1 *Let graph G of maximum degree d have a $(d+1)$-color recursive bisector of size $R(|G|)$ and bucket tree $B^{(b)}$. Let (C, h, c) be a cell tree in graph H, where $h = \lg(|G|/(bR(|G|))) + 1$ and $c = bR(|G|)$. Let C have cell slowdown p and f-transfer slowdown t. Then H can emulate G with slowdown $2(d^2+d)(\lceil 1/f \rceil)(2p+t)$, queue-size 1, and expansion $\lceil 2(|H|/|C|) \rceil$.*

We prove Theorem 1 by constructing the embedding of G into H, along with the specification of a routing regimen that achieves the claimed cost. First, we assign nodes of G to nodes of H.

Lemma 1 (Assignment Lemma) *Let G be a graph of maximum degree d, with a $(d+1)$-color recursive bisector of size $R(|G|)$ and bucket tree B. Let (C, h, c) be a cell tree in graph H, as in Theorem 1. Then nodes of G can be mapped into C so that nodes adjacent in G are mapped into cells at distance at most d apart, one of these cells being an ancestor of the other.*

Proof. By Proposition 2, G can be mapped to its bucket tree B so that every edge of G is stretched along a path between a node of B and its ancestor; no such path has more than d edges. We prove the Lemma by showing that bucket tree B can be mapped to cell tree C in a way that preserves the dilation of the embedding of G in its bucket tree B.

Recall that R is sublinear. By a simple inductive argument, one verifies that bucket capacities slowly decrease with level in the bucket tree. We have chosen b so that the cell capacity c equals the capacity of the root bucket B_λ, which is the largest in B.

The assignment of buckets of B to cells of C is as follows:

Step 0:
assign B_λ to C_λ;
assign B_0 to C_0;
assign B_1 to C_1.

Step k: $(1 \leq k < \lg|G|)$
{At this step, every bucket B_x has already been
assigned to some cell C_y, for $|y| \leq |x| = k$}.
If available capacity r of C_y suffices
to receive both B_{x0} and B_{x1}
then assign B_{x0} and B_{x1} to C_y;
else assign $\lfloor (r/2) \rfloor$ nodes of B_{x0} to C_y;
assign $\lfloor (r/2) \rfloor$ nodes of B_{x1} to C_y;
assign remaining nodes of B_{x0} to C_{y0};
assign remaining nodes of B_{x1} to C_{y1}.

The following observations establish the claim about the height h of the cell tree and the dilation of embedding G into it:

Since every bucket other than the root is smaller than the cell, no bucket can span more than two cells. Also, a child of a bucket can be only assigned either to the same cell as its parent, or to a child of the cell of its parent, or to both. At each step of the procedure, one level of the bucket tree is assigned, while at most one level of the cell tree is consumed. At most one node of C per bucket of B is left unoccupied, accounting for the factor of 2 in the expansion. □-Lemma 1

Our next task is defining the edge routing of the embedding. We simplify this task by appealing to the following well known result [18].

Proposition 3 *Every graph G of maximum degree d can be decomposed into at most $d+1$ partial subgraphs, each of maximum degree 1.*

Let $G_1, \ldots, G_{d+1}$ be the partial subgraphs resulting from decomposing G by Proposition 3. Our next step is computing the slowdown of emulating such a partial subgraph, say G_1, by graph H with its cell tree (C, h, c). The slowdown of emulating the entire graph G is then $(d+1)$ times greater, since the emulation proceeds in $(d+1)$-step rounds, one for each partial subgraph G_i.

Lemma 2 (Routing Lemma) *Let (C, h, c) be a cell tree in graph H, with cell slowdown p and f-transfer slowdown t. Let G_1 be a graph of maximum degree 1, whose nodes are assigned to C so that nodes adjacent in G_1 are assigned to cells at distance at most d apart, one of these cells being an ancestor of the other. Then H can emulate G_1 with slowdown $2d(\lceil 1/f \rceil)(2p+t)$ and queue-size 1.*

Proof. The specification of the routing has two parts. The first part, *macrorouting*, is global; it specifies only the *cells* that the routing paths visit. The second part, *microrouting*, is local; it specifies paths *within* cells. Our task is to define both parts of the routing and to schedule traversal of edges along the paths so as to justify the claimed cost. Our aim is to ensure that at every step, at each node, at most one message can be waiting to be communicated. Through macrorouting we guarantee that the *average* number of messages waiting at cell nodes is at most one, at any step. Through microrouting we ensure that there is at most one such message at any node any time.

Macrorouting. Given an edge e, let C_x and C_y be the cells (not necessarily distinct) to which the endpoints of e are assigned. Assume that C_x is m levels below C_y in the cell tree, where $0 \leq m \leq d$. (Note that $m = 0$ just when $C_x = C_y$.) We route e via a *macropath* comprising m *macrolinks*, each macrolink connecting two adjacent cells on the shortest path of m macrolinks, starting at the source cell C_x and ending at the destination cell C_y. The messages with origin in C_y and destination in C_x are routed along the same macropath, but they visit the macrolinks in the reversed order. The factor of 2 in the slowdown accounts for the two directions of traversal.

All edges of G_1 are macrorouted by a single orchestrated regimen. The orchestration mandates d *stages*. At stage j, where $1 \leq j \leq d$, active macrolinks are those that are incident in cells which are exactly $d-j$ macrolinks away from the destination cell of the macropath to which they belong. In other words, at stage 1 we cross the first macrolink of each macropath of length d; at stage 2 we cross the second macrolink in each such macropath, as well as the first link in each macropath of length $d-1$; at stage j, we cross the $(j-k)$th macrolink in each macropath of length $d-k$, where $0 \leq k < j$. Our macrorouting regimen, therefore, crosses macrolinks in the correct order.

Given a cell C_u, let the *macrocongestion* of C_u at stage j be the ratio of the number of macropaths departing from C_u at stage j and the number of nodes of C_u (which is the cell capacity c). To assess the macrocongestion of C_u at stage j, we appeal to the Assignment Lemma. It grants that the macropaths departing from C_u at stage j are just those macropaths that end at the ancestor cell of C_u which is exactly $d-j$ macrolinks above C_u; call this ancestor C_v. Since G_1 is of maximum degree 1, the maximum number of macropaths whose destination is C_v cannot exceed the cell capacity c of C_v; in particular, at most c such links can depart from C_u at step j. Thus the macrocongestion of any cell at any stage does not exceed one macropath per node of the cell.

Microrouting. First, we request that the macrolinks leaving cell C_u at stage j depart from f-transfer nodes of C_u and arrive at those nodes of the parent cell C'_u to which the f-transfer permutation maps f-transfer nodes of C_u. (Likewise, the macrolinks that arrive to cell C_u at stage $j-1$ end at nodes of C_u to which the f-transfer permutation maps f-transfer nodes of children cells of C_u.)

Since the macrocongestion of cell C_u at stage j is not greater than one, at most a total of c macropaths pass through C_u or start at C_u. Therefore, we can associate a node in C_u with each macropath that leaves C_u at stage j as follows: if there is a macropath of length $d-j+1$ starting at the node, then associate that macropath with the node; otherwise associate with the node one of the macropaths that pass through v. In such an association, call the node the *home* of the macropath. At stage d, choose for home nodes the very nodes on which macropaths end; note that home nodes chosen this way are guaranteed to be distinct from endpoints of edges whose both endpoints reside in the same cell, since G_1 is of maximum degree 1.

To arrange the microrouting with small congestion, we partition the nodes of cell C_u into (fc) *transfer groups*, of size $\lceil 1/f \rceil$ or $\lfloor 1/f \rfloor$ each, so that each group contains exactly one f-transfer node. Furthermore, we require that every group contains at most one node from each of the two disjoint sets of nodes to which the f-transfer permutation maps f-transfer nodes of each of two children cells. We then associate each group with the unique f-transfer node that it contains; call this node the *leader* of the group.

Now, let us focus on the first stage of the macrorouting, when $j=1$, in cell C_u. At the first stage all macropaths departing from C_u start at C_u; they therefore touch C_u at their home nodes. The microrouting sequence at this stage consists of $\lceil 1/f \rceil$ repetitions of a cycle of three elementary components: two spins and one transfer. A *spin* occurs by

routing paths *within* cells. A *transfer* occurs by routing paths *between* cells. The whole of $\lceil 1/f \rceil$ repetitions of this cycle are needed to remove congestion from f-transfer nodes:

1. First, we route one spin. This spin routes the paths from exactly one node in each transfer group to the leader of the group. Each spin defines a permutation on nodes of C_u by defining a transposition between a node of C_u and the leader of its group. This permutation obviously fixes cells setwise, so the cost of routing a spin in *all* cells simulteneously is the cell slowdown p.

2. After the first spin, we route one transfer. The transfer is the f-transfer permutation sending each leader to its matching node in the parent cell C'_u. The cost of one transfer in the *entire* cell tree simultaneously is the transfer slowdown t.

3. After the transfer, we route one more spin in the parent cell C'_u from destinations of the f-transfer permutation to home nodes of stage-2 macropaths. The cost of this spin in *all* cells simultaneously is again the cell slowdown p.

After $\lceil 1/f \rceil$ repetitions of this cycle of two spins and one transfer, all length-d macropaths have advanced one macrolink, so that the length of the longest portion of a macropath that has yet to be routed is $d-1$; moreover, the macropaths departing from every cell at stage 2 touch the cell at their distinct home nodes, thereby preparing the next-stage spin. A straightforward inductive extension verifies that the following invariant is maintained as the macropaths contract through all d stages: after stage j the macropaths reside with their distinct stage-$(j+1)$ home nodes; the length of the longest portion to be routed on any macropath is at most $d-j$ macrolinks. The very last spin accommodates those edges of G_1 whose both endpoints reside in a single cell, along with macropaths whose destinations are in that cell.

We arrive at the total emulation cost of $2d(\lceil 1/f \rceil)(2p+t)$ after combining the cost of d stages in each of the 2 directions, where each stage consists of $\lceil 1/f \rceil$ repetitions of the cycle of $2p+t$ steps. □-Lemma 2

The Assignment Lemma and the Routing Lemma justify the claim of Theorem 1.

Variations. In our approach, the only properties of the host graph that are visible to the guest graph are the host's cell tree parameters f, t, p. Indeed, for given cell size and height of the cell tree (which are determined by the bucket tree) a guest cannot distinguish between two hosts whose cell tree parameters are equal—both hosts provide emulations with identical slowdown. Since we have expressed emulation slowdown in terms of the three cell tree parameters, the only way to find efficient emulations by specific host networks is to strive for good values of those parameters (smaller p or t, or greater f). Later we indeed obtain such good values for two sample families of host networks; the resulting emulations are known [4, 3] to be existentially optimal across the class of bounded-degree separable graphs. The optimality of final cost leads us to believe that our choice of parameters is good, in that the parameters fully and faithfully represent the capability of host networks to emulate the class of arbitrary bounded-degree

separable graphs. We would like to know if our choice of representation parameters is the only good choice. Within the general strategy of our approach, other formulations are indeed possible; they may differ in inessential details, as long as they follow the same essential rules:

To build an efficient cell tree we ought to view the host node set as a collection of equal-size cells; we label the cells by nodes of the complete binary tree. Among all possible communication patterns in the host we are interested in two only. These can be visualized as mutually orthogonal: the "longitudinal" *transfer pattern* runs among cells and establishes fast routes between large subsets of nodes in adjacent cells; the "radial" *spin pattern* is confined within cell boundaries and connects cell nodes into constellations closely gathered around the nodes involved in transfer routes. At any phase in a cell tree emulation exactly one of these patterns occurs. Speeding up these two communication patterns amounts to speeding up the emulation—parameters that quantify them are a matter of choice.

Detailed solutions allow a lot of tuning. In the assignment stage, we note that the bisector size of the guest graph imposes only a lower bound on the size of cells; choosing larger cells affects slowdown and may improve expansion, but does not invalidate the assignment. In the macrorouting stage, we note that emulations can be speeded up if we allow messages to be queued up at the destination nodes of the f-transfer permutation. At the cost of maintaining queues of size $\lceil 1/f \rceil$ at these nodes, we can dispense with roughly half of the spins—those that route macropaths to their home nodes. Thus we could reduce the last factor in the slowdown of the Routing Lemma from $(2p+t)$ to $(p+t)$. Also, we observe that several efficient transfer permutations with various transfer rates may exist for a given host; moreover, a many-one mapping may be employed instead of the f-transfer permutation. In the microruting stage we could try to exploit the properties of spins to achieve them at cost smaller than that of an arbitrary cell permutation: some spins fix transfer groups (which are expected to be much smaller than cells) setwise; all spins are transpositions involving exactly one tranfer node in each pair.

We are now prepared to look for efficient cell trees in hypercubes and de Bruijn graphs. Revealing them in hypercubes is rendered simple by the hypercube's direct-product structure which yields an enriched tree with smaller hypercubes in its nodes. This straightforward construction still awards us a new embedding with improved expansion constant (Section 3). Good cell trees in de Bruijn graphs are less conspicuous; by discovering them we learn that de Bruijn graphs can successfully (and substantially better than so far known) emulate bounded-degree separable graphs (Section 4).

3 Application to Hypercubes

The *n-dimensional hypercube* $Q(n)$ is the graph whose nodes comprise the set Z_2^n and whose edges connect each node $x\beta y$ to node $x\overline{\beta}y$, where $\beta \in Z_2$ and $xy \in Z_2^{n-1}$.

Theorem 2 *Let graph G of maximum degree d have a $(d+1)$-color recursive bisector of size $R(|G|)$. Then G can be emulated by a hypecrcube with slowdown $O(d^2 \lg R(|G|))$, queue-size 1, and expansion 3.*

Proof. We have to find a cell tree (C, h, c) in the hypercube and to compute its slowdown factors.

Cells. Recall that order-n hypercube $Q(n)$ is a direct product $Q(h+1) \times Q(k)$ of two hypercubes whenever $n = (h+1)+k$. Choose k so that $c = 2^k$ nodes of $Q(k)$ are sufficient as cell capacity; this yields $k = O(\lg R(|G|))$, by the Assignment Lemma. Let each cell be an instance of $Q(k)$ in $Q(n) = Q(h+1) \times Q(k)$. Choose the height h so that the cell tree is large enough. It is well known (cf. [7]) that the complete binary tree $T(h)$ can be embedded in $Q(h+1)$ with dilation 2; a witnessing embedding assigns node x of $T(h)$ to node $x10^{h-|x|}$ of $Q(h+1)$. Label each cell, that is each instance of $Q(k)$ in $Q(n)$, by the name of the node of $T(h)$ which is assigned to the node of $Q(h+1)$ associated with that instance of $Q(k)$ in the product $Q(n) = Q(h+1) \times Q(k)$. We thus define the cell C_x as the following subset of nodes of $Q(n) = Q(h+1) \times Q(k)$.

$$C_x = \{(x10^{h-|x|}, y) \mid y \in Z_2^k\}$$

One node of $Q(h+1)$ remains unoccupied by this assignment, thus giving rise to the small increase in expansion over that coming from embedding the bucket tree into the cell tree C.

Given a non-root cell C_u, which is an instance of $Q(k)$ in $Q(n) = Q(h+1) \times Q(k)$, let C_{u_1} be the parent cell of C_u, so that $u = u_1\xi$, for some $\xi \in Z_2$.

Cell slowdown. Within each cell, that is within each instance of $Q(k)$ in $Q(n)$, cell permutations can be routed with cell slowdown $p = 2k - 1 = O(\lg R(|G|))$ (cf. [2]).

Transfer fraction. $f = 1/2$.

(1/2)-transfer nodes. In cell $C_{u_1\xi}$, we define the set $F(u_1\xi)$ of (1/2)-transfer nodes as the half of cell nodes which have ξ in the leftmost bit position of their second, k-bit component.

$$F(u_1\xi) = \{(u_1\xi 10^{h-|u_1|-1}, \xi z) \mid z \in Z_2^{k-1}\}$$

(1/2)-transfer groups in cell $C_{u_1\xi}$ consist of pairs of nodes, each pair comprising the leader $(u_1\xi 10^{h-|u_1|-1}, \xi z)$, for some $z \in Z_2^{k-1}$, and node $(u_1\xi 10^{h-|u_1|-1}, \bar{\xi} z)$, which differs from its leader only in the leftmost bit of its second, k-bit component.

(1/2)-transfer permutation. The (1/2)-transfer permutation is the map

$$(u_1\xi 10^{h-|u_1|-1}, \xi z) \mapsto (u_1 10^{h-|u_1|}, \xi z),$$

where $z \in Z_2^{k-1}$.

(1/2)-transfer slowdown. $Q(h+1)$ can emulate the tree $T(h)$ with slowdown 2; note that the (1/2)-transfer permutation is effected by traversing a subset of tree-edges of the embedded $T(h) \times Q(k)$, so $t = 2$ is the required (1/2)-transfer slowdown.

The claim follows from Theorem 1, instantiated with the f, p and t that we have just computed. □

The expansion of Theorem 2 should be compared with expansion-16 embedding of separable graphs into hypercubes, derived from their embedding into butterflies in [4, 3].

4 Application to de Bruijn Graphs

The *order-n de Bruijn graph* $D(n)$ [9] has node-set Z_2^n; each node βx, where $\beta \in Z_2$ and $x \in Z_2^{n-1}$, is connected by the *shuffle* edge to node $x\beta$, and by the *shuffle-exchange* edge to node $x\overline{\beta}$. Let $\mathcal{S}(\beta y) =_{\text{def}} y\beta$ and $\mathcal{E}(\beta y) =_{\text{def}} y\overline{\beta}$.

Theorem 3 *Let graph G of maximum degree d have a $(d+1)$-color recursive bisector of size $R(|G|)$. Then G can be emulated by a de Bruijn graph with slowdown $O(d^2 \lg R(|G|))$, queue-size 1, and expansion 3.*

Proof. Identifying a cell tree (C, h, c) in the de Bruijn graph is just a little bit more involved than in the hypercube. Consider a host de Bruijn graph of order $n = h + k + 1$.

Cells. Choose k so that $c = 2^k$ nodes are sufficient as cell capacity; this yields $k = O(\lg R(|G|))$, by the Assignment Lemma. Choose height h so that the cell tree is large enough. Given tree node $x \in Z_2^j$, $0 \leq j \leq h$, we define its cell as the following subset of $c = 2^k$ nodes of $D(n)$.

$$C_x = \{x^R 10^{n-k-1-j} y \mid y \in Z_2^k\}$$

where string x^R is the reversal of x. So, all $c = 2^k$ nodes in cell C_x are equal in their first $h + 1 = n - k$ bits, and have distinct k-bit suffixes. Note that c nodes of $D(n)$ of the form $0^{n-k}y$, $y \in Z_2^k$ do not belong to any cell, thus giving rise to the small increase in expansion over that imposed by embedding the bucket tree into the cell tree.

Cell slowdown. Every cell permutation in $D(n)$ can be routed with slowdown $p = 2k$ [1].

Transfer fraction. $f = 1/2$.

(1/2)-transfer nodes. In cell C_u, we define the set $F(u)$ of (1/2)-transfer nodes as the half of cell nodes which have 0 in the leftmost bit position of their k-bit suffix.

$$F(u) = \{u^R 10^{n-k-1-|u|} 0z \mid z \in Z_2^{k-1}\}$$

(1/2)-transfer groups in cell C_u consist of pairs of nodes, each pair comprising the leader $u^R 10^{n-k-1-|u|} 0z$, for some $z \in Z_2^{k-1}$, and node $u^R 10^{n-k-1-|u|} 1z$ which differs from its leader only in the leftmost bit of its k-bit suffix.

(1/2)-transfer permutation is the map $x \mapsto \mathcal{S}(x)$. To verify that it indeed takes (1/2)-transfer nodes to their parent cell, let $x \in F(u)$ be a (1/2)-transfer node in cell C_u

and let C_{u_1} be the parent cell of C_u, so that $u = u_1\xi$, $u^R = \xi u_1^R$, for some $\xi \in Z_2$. Note that the k-bit suffix of x is of the form $0z$ for some $z \in Z_2^{k-1}$. Then

$$x = u^R 10^{n-k-1-|u|}0z = \xi u_1^R 10^{n-k-1-|u|}0z$$

$$S(x) = u_1^R 10^{n-k-1-|u-1|}z\xi \in C_{u_1}.$$

(1/2)-transfer slowdown. Since the (1/2)-transfer permutation amounts to crossing shuffle edges, the (1/2)-transfer slowdown is $t = 1$.

The claim follows from Theorem 1, instantiated with the f, p and t that we have just computed. □

The expansion of Theorem 3 should be compared with expansion-16 embedding of separable graphs into butterflies in [4, 3].

Theorem 3 extends directly to hypercube-derivative relatives of de Bruijn graphs: to the closely related shuffle-exchange graphs (cf. [17]), and to product-shuffle graphs [15], as these graphs can emulate equal-sized de Bruijn graphs with slowdown 2.

5 Conclusion

We have presented the *cell tree* as an intermediate structure in a general technique for emulating arbitrary bounded-degree separable graphs. Its role in emulating the class of bounded-degree separable graphs is analogous to the role of the tree of meshes [8] in VLSI layouts of the same class of graphs. We have thus reduced the problem of emulating arbitrary bounded-degree separable graphs to the problem of emulating their generic host, the cell tree. By exhibiting efficient emulations of cell trees by hypercubes and de Bruijn graphs we have instantiated our technique for these two host families and obtained two new emulations of bounded-degree separable graphs. The emulations by de Bruijn graphs are exponentially faster than previously known, while the emulations by hypercubes achieve smaller expansion constants.

We are interested in the *structure* of underlying graphs of interconnection networks. For this particular problem we have succeeded in summarizing the relevant network properties using only a few *numerical* parameters: the cell tree parameters of the host graph and the bucket tree parameters of the guest graph. Our numerical representation is topology-independent and amenable to analysis and design, yet it captures those capabilities of parallel architectures that are of interest in our approach.

Acknowledgment. The author is greatly indebted to her research advisor Arnold L. Rosenberg for proposing the problem and for supplying many valuable comments and suggestions during the development of the solution and the preparation of the paper, and to Joyce Gastel and Antony L. Hosking for their helpful comments on the presentation.

References

[1] F.S. Annexstein (1989): Fault tolerance in hypercube-derivative networks. *1st ACM Symp. on Parallel Algorithms and Architectures*, 179-188.

[2] M. Baumslag and F.S. Annexstein (1990): A unified approach to global permutation routing on parallel networks. *2nd ACM Symp. on Parallel Algorithms and Architectures*, 398-406. *Math. Syst. Th.*, to appear.

[3] S.N. Bhatt, F.R.K. Chung, J.-W. Hong, F.T. Leighton, B. Obrenić, A.L. Rosenberg, E.J. Schwabe (1991): Optimal emulations by butterfly-like networks. Tech. Rpt. 90-108, Univ. Massachusetts; *J. ACM*, to appear.

[4] S.N. Bhatt, F.R.K. Chung, J.-W. Hong, F.T. Leighton, A.L. Rosenberg (1988): Optimal simulations by butterfly networks. *20th ACM Symp. on Theory of Computing*, 92-104.

[5] S.N. Bhatt, F.R.K. Chung, F.T. Leighton, A.L. Rosenberg (1986): Optimal simulations of tree machines. *27th IEEE Symp. on Foundations of Computer Science*, 274-282.

[6] S.N. Bhatt, F.R.K. Chung, F.T. Leighton, A.L. Rosenberg (1989): Universal graphs for bounded-degree trees and planar graphs. *SIAM J. Discrete Math. 2*, 145-155.

[7] S.N. Bhatt, F.R.K. Chung, F.T. Leighton, A.L. Rosenberg (1991): Efficient embeddings of trees in hypercubes. *SIAM J. Comput.*, to appear.

[8] S.N. Bhatt and F.T. Leighton (1984): A framework for solving VLSI graph layout problems. *J. Comp. Syst. Sci. 28*, 300-343.

[9] N.G. de Bruijn (1946): A combinatorial problem. *Proc. Koninklijke Nederlandsche Akademie van Wetenschappen (A) 49*, Part 2, 758-764.

[10] D.S. Greenberg, L.S. Heath and A.L. Rosenberg (1990): Optimal embeddings of butterfly-like graphs in the hypercube. *Math. Syst. Th. 23*, 61-77.

[11] R. Koch, F.T. Leighton, B. Maggs, S. Rao, A.L. Rosenberg, E.J. Schwabe (1990): Work-preserving emulations of fixed-connection networks. Submitted for publication; see also, *21st ACM Symp. on Theory of Computing*, 227-240.

[12] F.T. Leighton (1983): *Complexity Issues in VLSI: Optimal Layouts for the Shuffle-Exchange Graph and Other Networks.* MIT Press, Cambridge, Mass.

[13] F.T. Leighton, B. Maggs, S. Rao (1988): Universal packet routing algorithms. *29th IEEE Symp. on Foundations of Computer Science*, 256-269.

[14] A.L. Rosenberg (1981): Issues in the study of graph embeddings. In *Graph-Theoretic Concepts in Computer Science: Proceedings of the International Wkshp. WG80* (H. Noltemeier, ed.) *Lecture Notes in Computer Science 100*, Springer-Verlag, N.Y., 150-176.

[15] A.L. Rosenberg (1991): Product-shuffle networks: toward reconciling shuffles and butterflies. *Discr. Appl. Math.*, to appear.

[16] E.J. Schwabe (1991): *Efficient Embeddings and Simulations for Hypercubic Networks*. Ph.D. Thesis, MIT.

[17] H. Stone (1971): Parallel processing with the perfect shuffle. *IEEE Trans. Comp., C-20*, 153-161.

[18] V.G. Vizing (1964): On an estimate of the chromatic class of a p-graph (in Russian). *Diskret. Analiz 3*, 25-30.