Parallel Divide and Conquer on Meshes*

V. Lo
Computer Science Department
University of Oregon
Eugene, OR 97403-1202
lo@cs.uoregon.edu

S. Rajopadhye
IRISA
35042 Rennes, France
rajopadhye@irisa.fr

J. Telle, X. Zhong
Computer Science Department
University of Oregon
Eugene, OR 97403-1202

Abstract

We address the problem of mapping divide-and-conquer programs to mesh connected multicomputers with wormhole or store-and-forward routing. We propose the *binomial tree* as an efficient model of parallel divide-and-conquer and present two mappings of the binomial tree to the 2D mesh. Our mappings exploit regularity in the communication structure of the divide-and-conquer computation and are also sensitive to the underlying flow control scheme of the target architecture. We evaluate these mappings using new metrics which are extensions of the classical notions of dilation and contention. We introduce the notion of *communication slowdown* as a measure of the total communication overhead incurred by a parallel computation. We conclude that significant performance gains can be realized when the mapping is sensitive to the flow control scheme of the target architecture.

Keywords: mapping, embedding, divide and conquer algorithms, binomial tree, mesh connected machines, routing, wormhole routing, store and forward routing, contention, dilation.

*This research was supported by a grant from the Oregon Advanced Computing Institute (OACIS), and NSF Grants CCR-8808532, MIP-9108528, and MIP-8802454

1 Introduction

A well-known problem-solving paradigm that occurs in many computations is divide-and-conquer. If the subproblems are independent of each other, they may be executed in parallel, and this makes it a useful paradigm for designing large scale parallel programs. Divide-and-conquer has been studied by many researchers [7, 2, 3, 13, 14] and is applicable to a wide range of applications. In this paper, we address the problem of mapping degree-two divide and conquer computations on two-dimensional meshes. We represent these computations as a *binomial tree* and show that it runs in a phase by phase manner and that there is a regular pattern in the times at which messages are sent. In addition, the message volumes also exhibit regularity. One of our goals is to exploit all aspects of the regularity (topological, temporal, and message volume), and still develop parameterized mappings for a family of graphs rather than a single graph.

We present two mappings, called the *reflecting mapping* and the *growing mapping*, for embedding the divide-and-conquer binomial tree to a mesh. In addition to exploiting the regularity of the binomial tree, our mappings are sensitive to the flow-control technology of the target machine. Furthermore, we evaluate the mappings using new cost functions that are extensions of the standard contention and dilation metrics used in the embedding literature. We consider four pragmatic cases—whether the communication volumes are significantly larger or smaller than the startup overhead, and whether the routing mechanism is wormhole or store-and-forward.

The remainder of the paper is organized as follows: Section 2 discusses related work. Section 3 describes the regular structure of divide-and-conquer computations. We present our mappings in Section 4 and describe the new performance metrics in Section 5. Section 6 presents our analysis and we conclude with a discussion and indication of future work.

2 Related Work

The divide and conquer model has been widely recognized as an effective parallel programming paradigm [14, 7]. Recently, an algebraic theory has been developed to provide a general framework to describe this class of computations and the mapping of such class of algorithms to hypercube-like architectures has been also discussed by Mou and Hudak [13].

The "keep half, send half" strategy to improve the efficiency of a divide and conquer algorithm was observed in [7, 2]. However, the model is not formalized and the underlying binomial tree structure was not recognized in those papers. In [11], a spanning binomial tree of a hypercube was used to achieve efficient broadcasting. In [17], an H-tree embedding was proposed to embed a complete binary tree to a mesh. The embedding requires a slightly larger mesh.

Graph embedding techniques are usually used to emulate a machine of one topology by a machine of a different topology [15, 17]. The maximum dilation and contention (congestion) are used as the two major metrics to measure an embedding. In the context of mapping a parallel program which is modeled as a static task graph [16, 4], such an approach is usually used to assign tasks to processors [4]. However, as we have argued in this paper, that approach ignores the temporal aspect of a computation and the underlying communication technique. For example, it has been well recognized that in a wormhole-routed or a circuit switching multicomputer network, the distance effect, compared with the effect of the message collision, is negligible for the message latency [10, 8, 5].

Much work has been done to analytically model a network with different routing schemes [12, 8, 1]. However, these models are all based on some assumptions about the message distribution and thus are not accurate for a specific application. Our work complements these studies and gives important metrics in developing a mapping onto a network with store-and-forward or wormhole communication technology.

3 Divide and Conquer, and Binomial Trees

The traditional task graph structure for a degree-2 divide and conquer algorithm is a complete binary tree, $CBT(n)$ of $2^n - 1$ nodes (1 root and $2^{(n-1)}$ leaves). However, the CBT is not efficient since the computation proceeds from the root to the leaves and back up the tree in a level by level manner; so at any time only the nodes in a given level are active. As is well known in the folklore and noted by many authors [2, 7], an obvious way to improve the performance is to use the "keep half, send half" strategy, outlined below. The parameter α, used in step 2 below (we assume that $0 < \alpha \leq 1$), denotes the message size in each phase, as a fraction of the incoming message. Each node in the tree performs the following computation:

1. Receive a problem (of size x) from the parent (the host, if the node is the root).
2. Solve the problem locally (if 'small enough'), or divide the problem into two parts (each of size αx), and spawn a child process *to solve one of them.* In parallel, start solving the other one, by repeating Step 2.
3. Get the results from the children and combine them. Repeat until all children's results have been combined.
4. Send the results to the parent (the host, if the node is the root).

There are two stages in the computation—*divide* (Step 2), and *combine* (Step 3). Note that during the divide stage, each node receives a message from its parent exactly once, but may send messages multiple times (once to each child process that is spawned). During the combine stage, the message traffic is in the opposite direction—a node may receive many times, but sends exactly once. The patterns of data flow in the two stages are identical except for direction. We therefore restrict our analysis to the divide stage, without any loss of generality. We will also normalize our analysis so that the volume of the message sent to the root (the initial problem size) is unity.

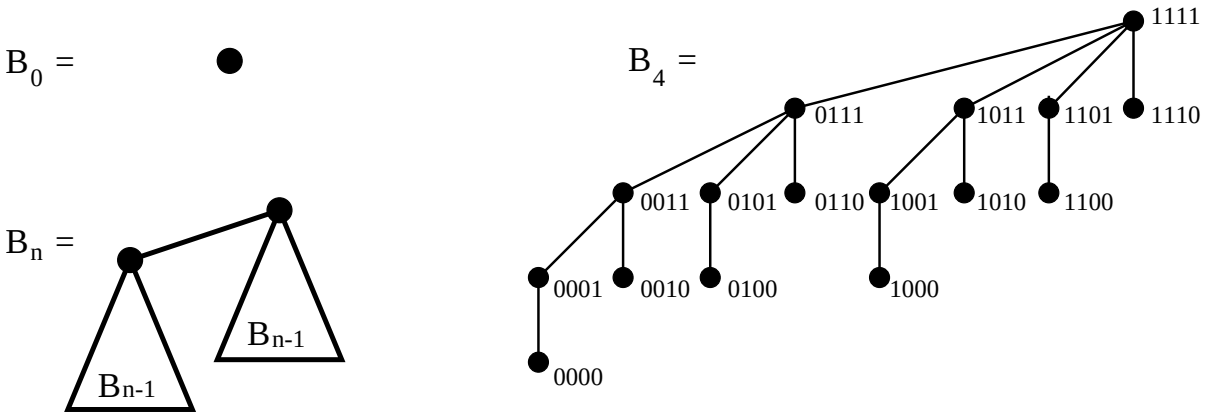


Figure 1: The binomial tree: definition, and example (B_4).

The parameter α described above can serve to completely define the communication volumes that occur in the divide and conquer paradigm. All edge weights can be expressed in terms of only α and the phase, i. Moreover, α is a natural parameter from the user's point of view. The two most common values are $\alpha = 1$ which corresponds to uniform traffic (this occurs in leader election, broadcasting a data value, and the combine stage of an associative-commutative operation), and $\alpha = \frac{1}{2}$, where the message volume is halved in each phase (this occurs in mergesort, data distribution, etc). There are also many problems where a different value may occur. A common example is solving a graph problem by decomposing the graph into two graphs each of half the number of vertices. Assuming messages passed to children consist of an adjacency matrix we have $\alpha = \frac{1}{4}$. Note that even for a single algorithm, the message volumes may be different in the divide stage and the combine stage. In multiplying a sequence of $n \times n$ matrices, for example, α is $\frac{1}{2}$ in the divide mode, and 1 in the combine mode. We now show that that the graph corresponding to the "keep half send half" strategy is a binomial tree.

Definition 1 The binomial tree $B(n)$ is defined inductively as follows [18] (see Fig. 1):

- $B(0)$ is a single node with no edges.
- $B(n)$ consists of two copies of $B(n-1)$ together with an edge connecting their roots, one of which is designated as the root of $B(n)$.

Note that the subtree rooted at any node is itself a binomial tree. Moreover, the root of $B(n)$ has n children, each of which are, in turn, the roots of $B(n-1), B(n-2), \ldots, B(0)$ (see Fig. 1). We will use the convention that the children of a node are arranged in decreasing order of size. Thus the i-th child of the root node is the root of $B(n-i)$. We adopt a postorder labelling of the tree as shown in Fig 1.

Lemma 1 The computation graph of the "keep half send half" paradigm divide and conquer with n divide phases is the binomial tree $B(n)$.

Proof: Let $C(n)$ be the graph corresponding to the "keep half send half" . $C(0)$ and $B(0)$ are both one-node graphs, hence identical. We establish the Lemma by showing that $C(n)$ and $B(n)$ have equivalent iterative definitions. By definition $C(n)$ is obtained by taking $C(n-1)$ and for each of its nodes creating a new node adjacent to it. We show by induction that $B(n)$ can be constructed from $B(n-1)$ in the same way. Since $B(0)$ is a single node and $B(1)$ consists of two nodes and a single edge, the base case holds. Assume inductively that $B(n)$ can be constructed by adding a leaf to every node of $B(n-1)$. By definition, $B(n)$ consists of two copies of $B(n-1)$, with an edge e connecting their roots. Adding a leaf to every node of $B(n)$ thus amounts to adding a leaf to every node in both copies of $B(n-1)$. By the inductive assumption this creates two copies of $B(n)$, with their roots still connected by the edge e, and the resulting graph corresponds to the definition of $B(n+1)$ ∎

In addition to the topological properties of the communication in divide and conquer algorithms as described above, it is important to accurately determine how it varies over time. In particular, we would like to specify exactly when a given edge is active, and what its weight is. Using the binomial tree $B(n)$ to model the computation, there are a total of n communication phases numbered 1 to n. As mentioned, we restrict attention to the divide stage of the computation. Every node receives a message exactly once. We say a node is activated after receiving this message, say at phase i. In each of the remaining phases $i+1, i+2, ..., n$ the node will itself activate a new child. Initially, the root receives a message of size 1 from the host, constituting the entire problem to be solved. In phase 1 the root sends a message of volume α to its first child, thereby activating it. In phase $1 \leq i \leq n$, there are 2^{i-1} active nodes, each sending a message of volume α^i to a child.

4 Mappings of the Binomial Tree to the 2-D Mesh

We present two mappings of $B(n)$ to the mesh of size $2^{\lfloor \frac{n}{2} \rfloor} \times 2^{\lceil \frac{n}{2} \rceil}$ (the mesh is either square or has an aspect ratio of two). Both mappings are 1:1 mappings, i.e., each node of the binomial tree is mapped to a distinct node of the mesh [1]. The first mapping uses Definition 1 of $B(n)$ and is called the Reflecting Mapping. The second mapping uses the definition of $B(n)$ which arises from Lemma 1 and is called the Growing Mapping. In both mappings, adjacent binomial tree nodes are mapped to mesh nodes in the same row or column. Thus, edges are mapped to the shortest path connecting their endpoints, and the mappings can be completely specified by the node mapping.

Definition 2 The reflecting mapping M_R is defined inductively as follows(see Figure 2):

- $B(0)$, the single node binomial tree is mapped to the single node mesh.
- If $n = 2k + 1$, $M_R(B(n))$ is constructed by taking two copies of $M_R(B(2k))$, and placing the second copy, reflected about a **vertical** axis, to the right of the first one. The roots of the two copies of $B(2k)$ (which must lie on the same row) are then connected. The root of the new (reflected) copy is the root of $B(n)$.
- If $n = 2k$, $M_R(B(n))$ is constructed by taking two copies of $M_R(B(2k-1))$, and placing the second copy, reflected about a **horizontal** axis, below the first one. The roots of the two copies of $B(2k-1)$ (which must lie on the same column) are then connected. The root of the reflected copy is the root of $B(n)$.

We will see that the reflecting mapping has excellent performance on machines with wormhole routing, but not so for store-and-forward networks. For this reason, we develop the *growing mapping* M_G. It uses the fact (shown in Lemma 1), that the binomial tree $B(n)$ can also be viewed as a copy of $B(n-1)$ to every node of which a new leaf node is added.

Definition 3 The growing mapping M_G is defined inductively as follows (see Figure 3):

[1] This restriction is justified since the size of the binomial tree can be controlled by adjusting the amount of work performed by the leaf nodes.

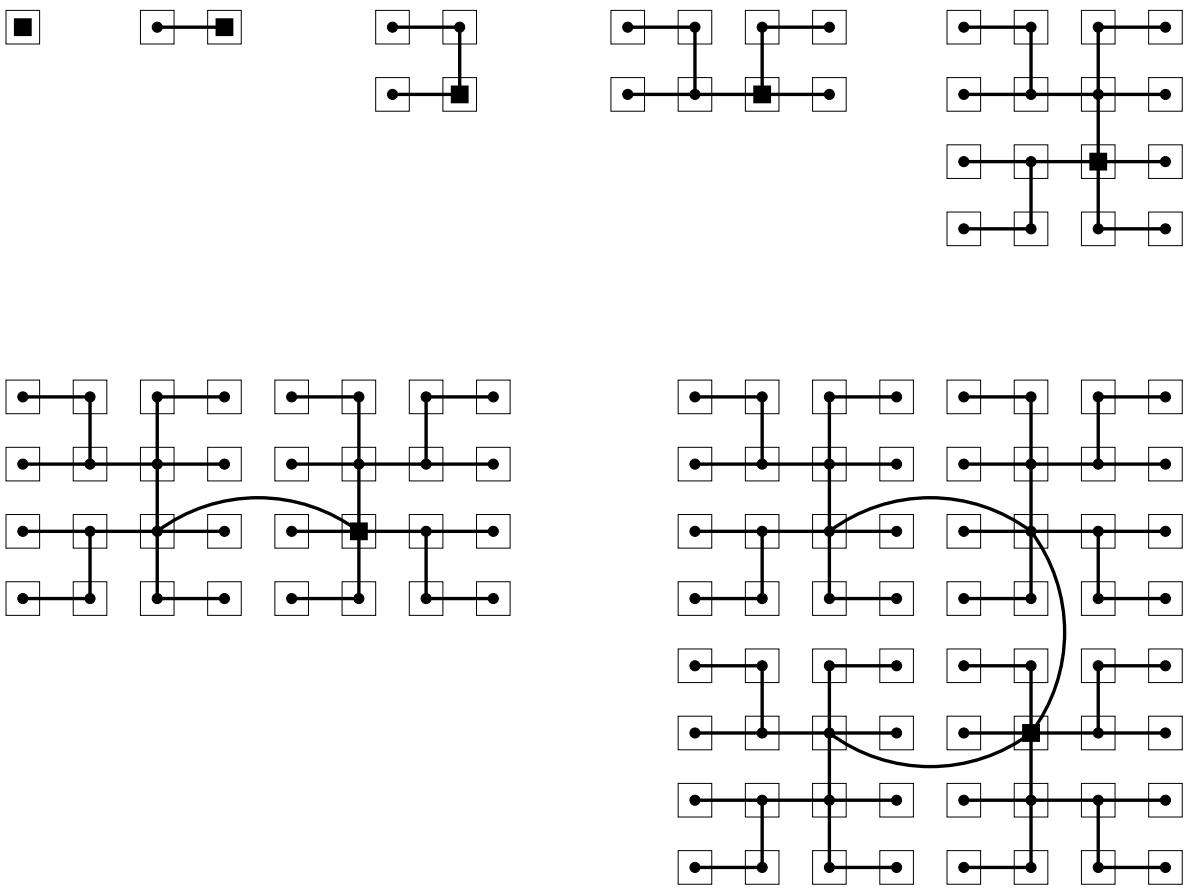


Figure 2: The reflecting mapping M_R (the square node is the root).

- For $B(0), B(1), B(2)$, M_G is the same as M_R.
- If $n = 2k-1$, non-leaf nodes are mapped as a $B(n-1)$ and leaf nodes are 'grown' horizontally from its parent with uniform dilation. Thus each node in the Eastern (Western) half of $B(n-1)$ gets its leaf placed in the same row as itself at a distance 2^{k-2} to the East (West).
- If $n = 2k$, non-leaf nodes are mapped as a $B(n-1)$ and leaves are 'grown' vertically from its parent with uniform dilation. Thus each node in the Northern (Southern) half of $B(n-1)$ gets its leaf placed in the same column as itself at a distance 2^{k-2} to the North (South).

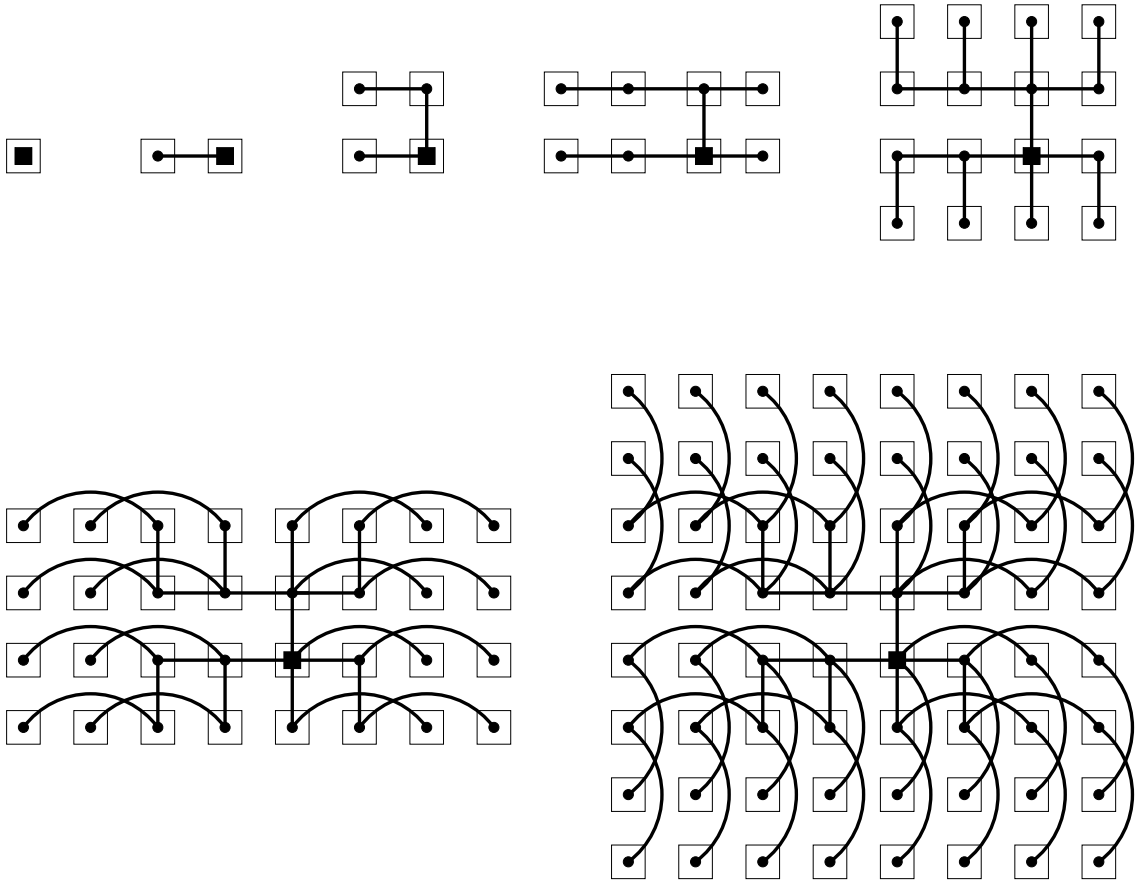


Figure 3: The growing mapping M_G (the square node is the root).

5 New Metrics for Performance Evaluation

In this section, we develop new metrics for evaluating a mapped parallel computation. Our metrics are extensions of the classic, graph theoretic metrics, congestion and dilation, taking into account (1) distinct message passing phases and (2) the volume of communication associated with each message. We also define the notion of *communication slowdown* which measures the amount of communication overhead incurred by a specific mapping relative to that incurred by the "perfect mapping."

We then derive formulæ for communication slowdown for two flow control schemes (wormhole and store-and-forward) and for two ranges of message volumes (when startup costs dominate, and when message volumes dominate). These formulæ are developed only for **contention free** mappings. Our new metrics assume two edges to contend only if they

are active in the same phase, a more realistic view than the classic graph theoretic notion of contention. We shall show that this holds for the growing mapping under store-and-forward routing, and for the reflecting mapping under both flow control schemes. General formulæ for arbitrary mappings is the subject of our ongoing work, and is beyond the scope of this paper.

The mapping problem typically uses the well known static task graph model of Stone [16] and Bokhari [4]. The parallel computation is viewed as a weighted graph, $G_C = \langle V_C, E_C, W \rangle$. The target machine is a graph $G_A = \langle V_A, E_A \rangle$, and a mapping, $\mathcal{M}$ is formally specified by the two functions $\mathcal{M}_n$ and $\mathcal{M}_e$ (the asterisk below denotes Kleene star).

$$\begin{aligned} \mathcal{M}_n &: V_C \rightarrow V_A \\ \mathcal{M}_e &: E_C \rightarrow V_A^* \end{aligned}$$

provided that for any $e = \langle a, b \rangle \in E_C$, $\mathcal{M}_e(e)$ is a path from $\mathcal{M}_n(a)$ to $\mathcal{M}_n(b)$. Typically, one seeks a mapping which minimizes the communication overhead while maintaining a balanced load. In the past, metrics that have been used to evaluate mappings use the following traditional notions of dilation and contention [15, 17]:

Definition 4 The **dilation**, $\mathcal{D}(e)$, **of an edge** $e \in E_C$, is $|\mathcal{M}_e(e)|$, i.e., the length of path, to which e is mapped. The **dilation**, $\mathcal{D}(E_C)$, of a mapping, $\mathcal{M}$, is given by $\mathcal{D}(E_C) = \max_{e \in E_C} \mathcal{D}(e)$.

Definition 5 The **contention**, $\mathcal{C}(l)$, **of a link** $l \in E_A$, is $|\{e \in E_C | l \in \mathcal{M}_e(e)\}|$, i.e., the number of edges in G_C that are mapped to paths containing l. The **contention**, $\mathcal{C}(E_C)$, of a mapping, $\mathcal{M}$, is given by $\mathcal{C}(\mathcal{M}) = \max_{l \in E_A} \mathcal{C}(l)$.

Dilation reflects the communication overhead caused by messages having to traverse multiple links, while contention reflects the communication overhead that arises when two or more messages require the same link.

As mentioned above, we are interested in phased computations which operate as follows. In the i-th phase, only the nodes in a subset, V_i, of V_C perform a computation, and messages are sent only on edges that belong to $E_i \subseteq E_C$. We assume a loosely synchronous model where the tasks are assumed to synchronize after each phase. We call this a **phased** computation graph, and a special case is a **uniform phased** graph, where the edge weights are uniform. We now refine the definitions of dilation to more realistically reflect (1) phasewise behavior, and (2) non-uniform edge weights. We will define functions of either an *individual* edge e (which is active in the i-th phase), or of *all* edges in the i-th phase, E_i. As a guide to the notation, a function of a *set* of edges is the *maximum* of its value over all elements of the set. For example, $W(e)$ is the weight of a single edge and $W(E_i)$ is the weight of the heaviest edge in the i-th phase.

Definition 6 The **i-th phase interference set**, $I(e)$ of an edge $e \in E_i$, consists of all the other edges in E_i which also use some link that is used by e, i.e., $I(e) = \{e' \in E_i \mid e \neq e' \wedge \mathcal{M}_e(e) \cap \mathcal{M}_e(e') \neq \emptyset\}$. "Path level contention," as introduced by Chittor [6] is a similar notion, except he does not deal with phased computations.

Definition 7 For an edge e, its **weighted dilation**, $\mathcal{D}_w(e)$, is the product of its weight and the length of the path to which it is mapped, i.e., $\mathcal{D}_w(e) = W(e) \times \mathcal{D}(e)$.
For a mapping, $\mathcal{M}$, the **i-th phase weighted dilation** is defined as the maximum weighted dilation of all the edges in that phase: $\mathcal{D}_w(E_i) = \max_{e \in E_i} \mathcal{D}_w(e)$.
For an edge $e \in E_i$, its **i-th phase weighted contention**, $\mathcal{C}_w(e)$, is the sum of the weights of edges in its i-th phase interference set, $\mathcal{C}_w(e) = \sum_{e' \in I(e)} W(e')$.
For a mapping, $\mathcal{M}$, its **i-th phase weighted contention**, $\mathcal{C}_w(E_i)$, is defined as the maximum i-th phase weighted contention of the edges in that phase, $\mathcal{C}_w(E_i) = \max_{e \in E_i} \mathcal{C}_w(e)$.

A special case arises when the computation is uniform, (i.e., $W(e) = 1, \forall e \in E_C$. Then, $\mathcal{D}_w(E_i) = \max_{e \in E_i} \mathcal{D}(e)$. Also, $\mathcal{C}_w(e) = |I(e)|$ and $\mathcal{C}_w(E_i) = \max_{e \in E_i} |I(e)|$. We indicate these **uniform** metrics by dropping the subscript w.

Definition 8 Let $\mathcal{T}(e)$ denote the **communication time for an edge**, $e \in E_i$, accounting for delays due to the path length and congestion.
The **communication time for the i-th phase**, $\mathcal{T}(E_i)$ is the largest time of all edges in

the phase (remember that our model assumes synchronization between the phases) i.e., $\mathcal{T}(E_i) = \max_{e \in E_i} \mathcal{T}(e)$.
The **total communication time** for the computation $\mathcal{T}(E_C)$ is the sum over all the phases, i.e., $\mathcal{T}(E_C) = \sum_i \mathcal{T}(E_i) = \sum_i \max_{e \in E_i} \mathcal{T}(e)$.

Definition 9 A mapping, $\mathcal{P}$, is said to be **perfect** if it has minimum dilation and minimum contention.

Note that both dilation and contention are minimized if the task graph G_C is a subgraph of the target G_A. This gives us a lower bound on the optimal cost of any mapping.

We define the notion of communication slowdown by normalizing the total communication time for a mapping with respect to the total communication time for the perfect mapping.

Definition 10 Given a mapping, $\mathcal{M}$, its **communication slowdown**, $\mathcal{S}(\mathcal{M})$ is defined as the ratio between its total communication cost and the cost of the perfect mapping, i.e., $\mathcal{S}(\mathcal{M}) = \frac{\mathcal{T}_{\mathcal{M}}(E_C)}{\mathcal{T}_{\mathcal{P}}(E_C)}$

5.1 Formulæ for Communication Slowdown

We now derive formulæ for the total communication time $\mathcal{T}(E_C)$ of a mapped computation. Recall that for this paper, we make the simplifying assumption that there is no contention. We will show in section 6 that this assumption is valid for our analysis. As mentioned earlier, we consider two different flow control schemes in the target machine, namely wormhole routing and store-and-forward routing. We will also consider two cases of message traffic: large volume (message size is the dominant factor) and small volume (where startup costs dominate).

5.1.1 Wormhole Routing

Consider a single edge, $e \in E_C$, in isolation. Under wormhole routing, a message consists of a stream of *flits* that are routed through the network with minimal buffering in a pipelined fashion. Thus, *in the absence of contention*, the time a message takes to travel from one processor to another over a path of length d is given in [11] as

$$\mathcal{T}(e) = c + b(W(e) + dh) \tag{1}$$

Here, c is the startup cost, $1/b$ is the bandwidth, and h is the flit size (usually, 1 or 2 bytes). Now, for large messages, $W(e) \gg dh$ and $bW(e) \gg c$, so we have

$$\mathcal{T}(e) \approx bW(e) \tag{2}$$

On the other hand, for small messages, $bW(e) \ll c + bdh$, and in this case, it is also reasonable to assume that $c \gg bdh$ (note that this term corresponds to an "additional" volume of dh, and this is of the order of a few tens of bytes, the same order of magnitude as the smallest messages in the system), and we have

$$\mathcal{T}(e) \approx c \tag{3}$$

Note that the distance is no longer relevant; this has been observed by [9]. Now, the estimated communication time for the i-th phase is

$$\mathcal{T}(E_i) = \max_{e \in E_i} \mathcal{T}(e) \approx \begin{cases} c & \text{small volume} \\ bW(E_i) & \text{large volume} \end{cases}$$

where $W(E_i)$ is the weight of the heaviest edge in phase i. Thus, the total communication time for a mapping with k phases, assuming no contention, is as follows:

$$\mathcal{T}(E_C) = \sum_{i=1}^{k} \mathcal{T}(E_i) \approx \begin{cases} ck & \text{small volume} \\ b \sum_{i=1}^{k} W(E_i) & \text{large volume} \end{cases} \tag{4}$$

Note that this formula applies to any contention free mapping, including the perfect mapping. Thus, we have the communication slowdown

$$\boxed{\mathcal{S}(\mathcal{M}) = \frac{\mathcal{T}_{\mathcal{M}}(E_C)}{\mathcal{T}_{\mathcal{P}}(E_C)} \approx 1 \qquad \text{small/large volume}} \tag{5}$$

for any contention free mapping $\mathcal{M}$ to a target machine with wormhole routing.

5.1.2 Store and Forward Routing

With store-and-forward routing, each message is decomposed into a sequence of messages, one for each "hop" along the path. Thus, *in the absence of contention*, the communication time for a single message to traverse a distance d is $\mathcal{T}(e) = d(c + bW(e))$, where c is the per node overhead cost, and $1/b$ is the bandwidth. We have the following extreme cases:

$$\mathcal{T}(e) \approx \begin{cases} c\mathcal{D}(e) & \text{small volume} \\ b\mathcal{D}(e)W(e) & \text{large volume} \end{cases} = \begin{cases} c\mathcal{D}(e) & \text{small volume} \\ b\mathcal{D}_w(e) & \text{large volume} \end{cases} \tag{6}$$

As before we can obtain the communication time for the mapping as follows:

$$\mathcal{T}(E_i) \approx \begin{cases} c \max\limits_{e \in E_i} \mathcal{D}(e) & \text{small volume} \\ b \max\limits_{e \in E_i} \mathcal{D}_w(e) & \text{large volume} \end{cases} = \begin{cases} c\mathcal{D}(E_i) & \text{small volume} \\ b\mathcal{D}_w(E_i) & \text{large volume} \end{cases} \tag{7}$$

$$\mathcal{T}(E_C) \approx \begin{cases} c \sum\limits_{i=1}^{k} \mathcal{D}(E_i) & \text{small volume} \\ b \sum\limits_{i=1}^{k} \mathcal{D}_w(E_i) & \text{large volume} \end{cases} \tag{8}$$

Note that the conventional dilation metrics are used in a phasewise additive sense. For the perfect mapping, the dilation is always 1, and we have

$$\mathcal{T}_{\mathcal{P}}(E_C) \approx \begin{cases} ck & \text{small volume} \\ b \sum\limits_{i=1}^{k} W(E_i) & \text{large volume} \end{cases} \tag{9}$$

Thus the communication slowdown is

$$\mathcal{S}(\mathcal{M}) \approx \begin{cases} \frac{1}{k}\sum_{i=1}^{k} \mathcal{D}(E_i) & \text{small volume} \\ \frac{\sum_{i=1}^{k} \mathcal{D}_w(E_i)}{\sum_{i=1}^{k} W(E_i)} & \text{large volume} \end{cases} \tag{10}$$

for any contention free mapping M to a target machine with store-and-forward routing.

6 Performance Analysis

We now analyze the two mappings presented in Sec 4 with respect to our new metrics. We will first show that the *reflecting mapping* is optimal for wormhole routing. (This is true because the mapping is contention free in the sense that no two edges in the same phase are simultaneously active.) We also show that the growing mapping is also contention free with store-and-forward routing. (In this case, there is no contention because the communication is such that it can be pipelined in a lock-step manner). Hence the above formulae are applicable. We compare the communication slowdown metrics of both mappings for store-and-forward routing, and show that the *growing mapping* has superior performance.

6.1 The reflecting mapping with wormhole routing

Lemma 2 For the reflecting mapping $M_R(B(n))$, edges using the same link are never active in the same phase. Thus, $\mathcal{C}_w(E_i) = 0$ for all phases $1 \leq i \leq n$ and $M_R(B(n))$ is contention free.

Proof: We prove this by showing (by induction on n) that edges using the same link are never active in the same phase.

$B(0)$ has no communication. Assume the claim holds for $0 \leq i < n$. By definition, $B(n)$ consists of two copies of $B(n-1)$ and an edge e connecting the roots of the two copies. Under M_R no two edges from separate copies of $B(n-1)$ share any links, hence by the inductive assumption the claim holds for all edges in these copies. The edge e is active only in the first phase and it is the only active edge in that phase. Hence the claim holds for $B(n)$. ∎

Theorem 1 For the reflecting mapping with wormhole routing, $M_R(B(n))$, the communication slowdown $\mathcal{S}(M_R) = 1$. Thus, the reflecting mapping is optimal.

Proof: This fact follows directly from Lemma 2 and Equation 5 for communication slowdown for a contention free mapping to a wormhole machine. ∎

6.2 The reflecting mapping with store-and-forward routing

In this section and the next, we omit the two constants c and b from the formulas for the total communication time of a mapping (see Equation 8). Both are multiplicative constants which ultimately cancel out in the formulas for communication slowdown.

We first establish the maximum weighted dilation $\mathcal{D}_w(E_i)$ for each phase.

Lemma 3 The maximum weighted dilation for the reflecting mapping in phase i is

$$\mathcal{D}_w(E_i) = \left(\frac{2^{\lceil \frac{i}{2} \rceil} - (-1)^{\lceil \frac{i}{2} \rceil}}{3} \right) \alpha^i$$

Proof: We have already argued that weights of all edges in phase i is α^i. Note that $\mathcal{D}_w(E_i) = \mathcal{D}(E_i)\alpha^i$ and that the regularity of the mapping implies that $\mathcal{D}(E_i)$ represents the dilation of any edge in phase i. Consider Fig 4 which shows the reflecting mapping of $B(2k)$ to a $2^k \times 2^k$ mesh, with $B(2k)$ rooted at I and its largest subtree $B(2k-1)$ rooted at F. Note that IF = CF = $\mathcal{D}(E_{2k})$ and BC = $\mathcal{D}(E_{2k-2})$, because the alternate horizontal and vertical reflections give $\mathcal{D}(E_{2k}) = \mathcal{D}(E_{2k-1})$. We also have AD = $2^{k-1} - 1$ and DE = 1, so that AE = 2^{k-1}. Our intermediate goal is to show the recurrence

$$\mathcal{D}(E_{2k}) = 2^{k-1} - \mathcal{D}(E_{2k-2}) \tag{11}$$

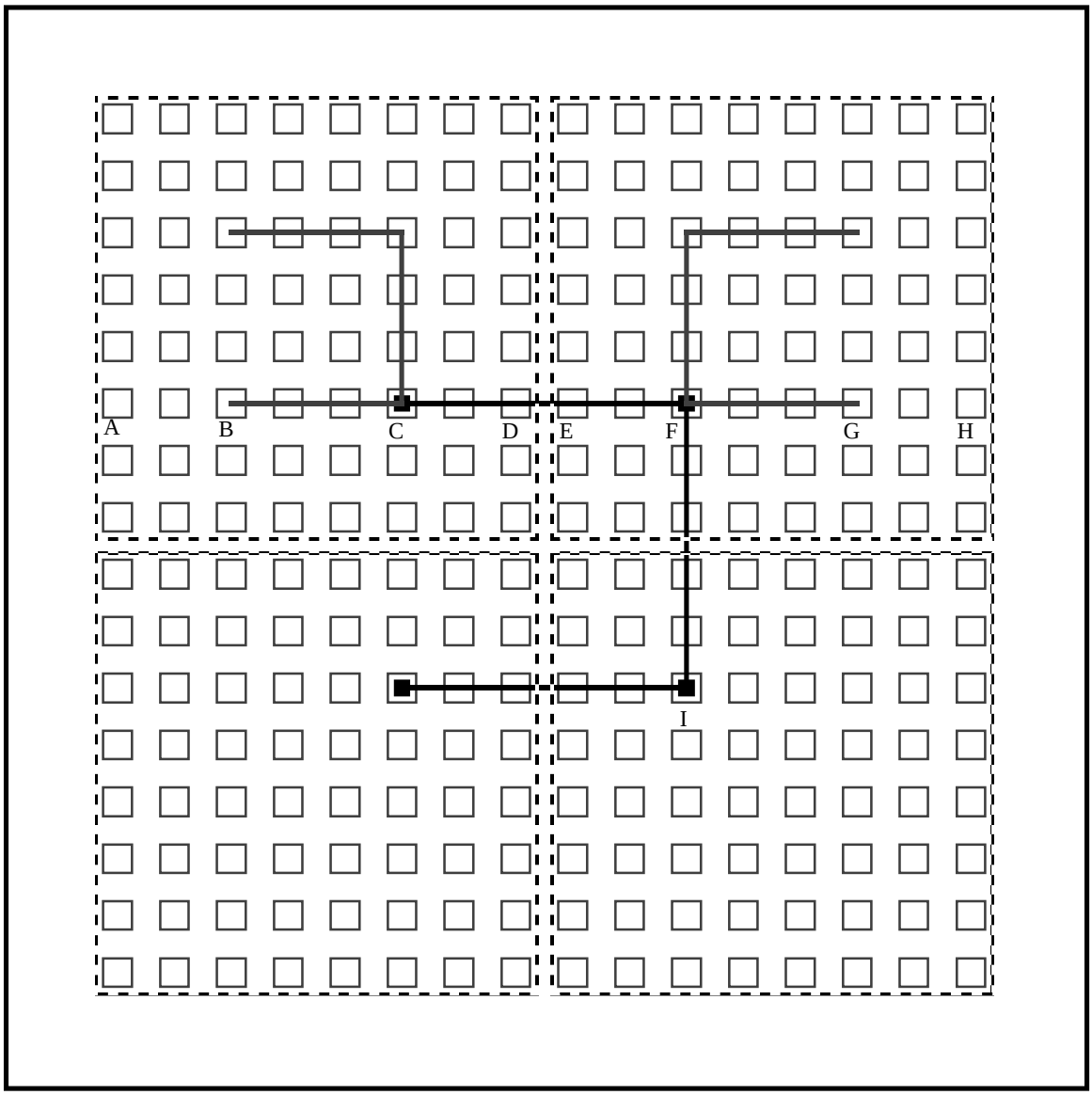


Figure 4: Mapping M_R: relation between $\mathcal{D}_w(E_{2k})$, $\mathcal{D}_w(E_{2k-1})$ and $\mathcal{D}_w(E_{2k-2})$

which, from the above, corresponds to $CF = AE - BC$. By observation, $CF = BF - BC$, so it only remains to show $AE = BF$. It can be easily shown that the root of $B(2k)$ is always mapped to the main diagonal of the mesh (top-left to bottom-right), and the root of $B(2k-1)$ is on the other diagonal (bottom-left to top-right). Thus, we have AB = CD and by the symmetry of reflection CD = EF, so that AB = EF. The last equality, together with an easy geometrical argument, gives AE = BF, as desired.

Now, the solution to the recurrence 11 (using standard textbook techniques) when multiplied by α^i establishes the Lemma for even i. For odd i, we have already argued that $\mathcal{D}(E_{2k-1}) = \mathcal{D}(E_{2k})$. ∎

Lemma 4 The communication time for the perfect mapping $\mathcal{T}_{\mathcal{P}}(E_C)$, follows directly from the fact that the edge weights of the binomial tree constitute a geometric series.

$$\mathcal{T}_{\mathcal{P}}(E_C) = \sum_{i=1}^{n} W(E_i) = \sum_{i=1}^{n} \alpha^i = \begin{cases} n & \text{if } \alpha = 1 \\ \frac{1-\alpha^{n+1}}{1-\alpha} - 1 & \text{otherwise} \end{cases} \tag{12}$$

$\mathcal{T}_{M_R}(E_C) = \sum_i \mathcal{D}_w(E_i)$ is also the sum of a geometric series whose solution is summarized in the following lemma, with focus on the common cases $\alpha = 1$ and $\alpha = \frac{1}{2}$.

Lemma 5 The total communication time for the reflecting mapping of $B(2k)$ on store-and-forward machines is given by

(a). $\mathcal{T}_{M_R}(E_C) = c_1 2^k + c_2(-1)^{k+1} + c_3\alpha^{2k}$ for $0 < \alpha \leq 1$

where $c_1 = \frac{2\alpha(1+\alpha)}{3(2-\alpha^2)}$, $c_2 = \frac{\alpha(1+\alpha)}{3(1+\alpha^2)}$, $c_3 = \frac{\alpha^3(1+\alpha)}{(1+\alpha^2)(\alpha^2-2)}$.

(b). $\mathcal{T}_{M_R}(E_C) = \frac{4}{3}2^k + c_4$ for $\alpha = 1$

where $c_4 = -2/3$ or $c_4 = -4/3$ for k odd or even, respectively.

(c). $\mathcal{T}_{M_R}(E_C) = \Theta(2^k)$ for $\alpha = \frac{1}{2}$

Substituting this and Equation 12 in Equation 10, we have the following.

Theorem 2 With store-and-forward routing the reflecting mapping of $B(2k)$ has slowdown

- $\mathcal{S}(M_R) \geq 2^{k-1}/k$ for $\alpha = 1$, large volume case
- $\mathcal{S}(M_R) = \Theta(2^k/k)$ for $0 < \alpha \leq 1$, large volume case
- $\mathcal{S}(M_R) \geq 2^{k-1}/k$ for $0 < \alpha \leq 1$, small volume case

6.3 The growing mapping under store-and-forward routing

We first establish the maximum weighted dilation $\mathcal{D}_w(E_i)$ for the growing mapping M_G. We will also show that it is contention free.

Lemma 6 In phases 1 and 2, M_G has weighted dilation α and α^2, respectively. For phase $i \geq 3$ we have $\mathcal{D}_w(E_i) = \alpha^i(2^{\lceil \frac{i}{2} \rceil - 2})$.

Proof: From the definition of M_G, we have dilation = 1 for phases 1 and 2, for phase $i \geq 3$, we have the dilation of each edge $2^{\lceil \frac{i}{2} \rceil - 2}$ and as before the volume of each edge α^i. ∎

Lemma 7 With store-and-forward, the growing mapping has no contention, since for each phase i, $\mathcal{C}_w(E_i) = 0$.

Proof: In phases 1, 2, 3 and 4 messages of M_G have dilation = 1 and thus clearly no contention. Because all messages in the i-th phase for $i > 4$ have the *same* volume, no two messages will compete for the same link simultaneously. To see this, consider a single message and its interference set, which are routed along a single row or column of the mesh (see dotted edges in Figure 3). In lockstep fashion, all messages in this set are sent without any interference, in $2^{\lceil \frac{i}{2} \rceil - 2}$ steps, with every message moving one hop closer to its destination in each step. Thus the contention for the growing mapping in the store-and-forward case is $\mathcal{C}_w(E_i) = 0$. ∎

For the reflecting mapping of $B(2k)$, we have $\mathcal{T}_{M_G}(E_C) = \sum_i \mathcal{D}_w(E_i) = \alpha + \alpha^2 + \sum_{3 \leq i \leq 2k} \alpha^i(2^{\lceil \frac{i}{2} \rceil - 2})$, whose solution is summarized in the next lemma.

Lemma 8 The total communication time for the growing mapping of $B(2k)$ on store-and-forward machines is given by

(a). $\mathcal{T}_{M_G}(E_C) = c_5(2\alpha^2)^k + c_6$ for $0 < \alpha \leq 1$ but $\alpha \neq \frac{1}{\sqrt{2}}$

where $c_5 = \frac{\alpha(1+\alpha)}{2(2\alpha^2-1)}$ and $c_6 = \frac{\alpha(\alpha^3+\alpha^2-\alpha-1)}{2\alpha^2-1}$

(b). $\mathcal{T}_{M_G}(E_C) = 2^k$ for $\alpha = 1$

(c). $\mathcal{T}_{M_G}(E_C) = 1.125 - \frac{3}{2^{k+2}}$ for $\alpha = \frac{1}{2}$

(d). $\mathcal{T}_{M_G}(E_C) = \frac{\sqrt{2}+1}{4}(k+1)$ for $\alpha = \frac{1}{\sqrt{2}}$, for the case where the denominator in (a) is 0.

Substituting this and Equation 12 in Equation 10, we have the following.

Theorem 3 With store-and-forward routing the growing mapping of $B(2k)$ has slowdown:

- $\mathcal{S}(M_G) = 2^{k-1}/k$ for $\alpha = 1$, large volume case.
- $\mathcal{S}(M_G) = \mathcal{O}(1)$, assymptotically approaching 1.125, for $\alpha = 0.5$, large volume case.
- $\mathcal{S}(M_G) = 2^{k-1}/k$ for $\alpha = 1$, small volume case.

6.4 Comparison

We compare our two mappings in the following theorem, which follows from comparing the Lemmas and Theorems from the preceding sections.

Theorem 4

Wormhole routing For any α the reflecting mapping is optimal, whereas the growing mapping has some contention for each phase $i > 4$.

Store-and-forward routing For both small and large volume cases, the growing mapping of $B(2k)$ outperforms the reflecting mapping of $B(2k)$ for any $0 < \alpha \leq 1$ and any $k > 2$. For $k \leq 2$ both mappings are optimal.

With store-and-forward routing, both for the small volume case, any α, and the large volume case, $\alpha = 1$ (*i.e.* uniform communication), the growing mapping has 25% less communication time than the reflecting mapping, as the number of phases increases. With store-and-forward routing, the large volume case, $\alpha = 0.5$, the growing mapping outperforms the reflecting mapping by a factor which is exponential in the number of phases.

7 Conclusion

In this paper we have presented two algorithms for mapping the binomial tree divide-and-conquer computation to the 2-D mesh. These mapping algorithms exploit regularity

in the topological communication structure of the binomial tree as well as regularity in the communication phases of the divide and conquer binomial tree. In addition, our mapping algorithms are sensitive to the topological structure of the 2-D mesh and to the underlying flow-control scheme supported by the target machine (store-and-forward routing versus wormhole routing). We have developed a new performance metric called *communication slowdown* for evaluation of the communication overhead incurred when a phased computation is mapped to multicomputers. Evaluation of our two mappings with respect to communication slowdown shows that the reflecting mapping is optimal for wormhole routing, while the growing mapping is close to optimal for a wide range of message volumes and mesh sizes.

Our ongoing work in this area includes validation of our analysis for the reflecting and growing mappings through simulation; extensions for the case of medium size messages and for computations that have different α values in the divide and the conquer phases, respectively; and development and validation of our completion time and communication slowdown metrics that are applicable to arbitrary mappings of a wider range of parallel computations.

References

[1] A. Agarwal. *Limits on Network Performance.* MIT VLSI Memo, 1990.

[2] W. C. Athas and C. L. Seitz. Multicomputers: Message-passing concurrent computers. *IEEE Computer*, pages 9–23, August 1988.

[3] G. E. Blelloch. *Vector Models For Data-Parallel Computing.* THE MIT PRESS, 1990.

[4] S. H. Bokhari. *Assignment Problems in Parallel and Distributed Computing.* Kluwer Academic Publishers, 1987.

[5] S.H. Bokhari. Communication overhead on the Intel iPSC-860 hypercube. Technical report, ICASE, NASA Langley Research Center, May 1990.

[6] S.S. Chittor. *Communication Performance of Multicomputers.* PhD thesis, Dept. of Computer Science, Michigan State University, 1991.

[7] M. I. Cole. *Algorithmic Skeletons: Structured Management of Parallel Computation.* In *Research Monographs in Parallel and Distributed Computing.* MIT Press, 1989.

[8] W.J. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE Trans. Computers.*, C-39(6):775–785, June 1990.

[9] T.H. Dunigan. Performance of the Intel iPSC/860 and Ncube 6400 hypercubes. *Parallel Computing*, 17:1285–1302, 1991.

[10] C.L. Seitz et al. The performance and programming of the Ametek series 2010 multicomputer. In *Hypercube Computer Conference*, pages 33–36, 1988.

[11] S.L. Johnson. Communication in network architectures. In *VLSI and Parallel Computation*, page 290. Morgan Kaufmann Publishers, Inc., 1990.

[12] P. Kermani and L. Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks*, 3:267–286, 1979.

[13] Z.G. Mou and P. Hudak. An algebraic model for divide-and-conquer algorithms and its parallelism. *The Journal of Supercomputing*, 2(3):257–278, November 1988.

[14] P. A. Nelson and L. Snyder. Programming paradigms for nonshared memory parallel computers. In *The Characteristics of Parallel Algorithms*, pages 3–20. The MIT Press, 1987.

[15] A. L. Rosenberg. Graph embeddings 1988: Recent breakthroughs new directions. Technical Report 88-28, University of Massachusetts at Amherst, March 1988.

[16] H. S. Stone. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Transactions on Software Engineering*, SE-3(1):85–93, January 1977.

[17] J.D. Ullman. *Computational Aspects of VLSI.* Computer Science Press, Rockville, MD, 1984.

[18] J. Vuillemin. A data structure for manipulating priority queues. *Communications of the ACM*, 21(4):309–315, April 1987.

Abstract

We address the problem of mapping divide-and-conquer programs to mesh connected multicomputers with wormhole or store-and-forward routing. We propose the *binomial tree* as an efficient model of parallel divide-and-conquer and present two mappings of the binomial tree to the 2D mesh. Our mappings exploit regularity in the communication structure of the divide-and-conquer computation and are also sensitive to the underlying flow control scheme of the target architecture. We evaluate these mappings using new metrics which are extensions of the classical notions of dilation and contention. We introduce the notion of *communication slowdown* as a measure of the total communication overhead incurred by a parallel computation. We conclude that significant performance gains can be realized when the mapping is sensitive to the flow control scheme of the target architecture.

Keywords

mapping, embedding, divide and conquer algorithms, binomial tree, mesh connected machines, routing, wormhole routing, store and forward routing, contention, dilation.

Figure Captions

Figure 1: The binomial tree: definition, and example (B_4).

Figure 2: The reflecting mapping M_R (the square node is the root).

Figure 3: The growing mapping M_G (the square node is the root).

Figure 4: Mapping M_R: relation between $\mathcal{D}_w(E_{2k})$, $\mathcal{D}_w(E_{2k-1})$ and $\mathcal{D}_w(E_{2k-2})$

Preferred Address

Virginia Lo

Computer Science Department

University of Oregon

Eugene, OR 97403-1202

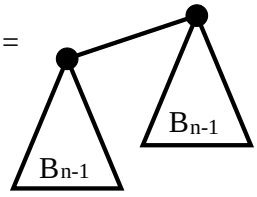
lo@cs.uoregon.edu

503-346-4473 (ph)

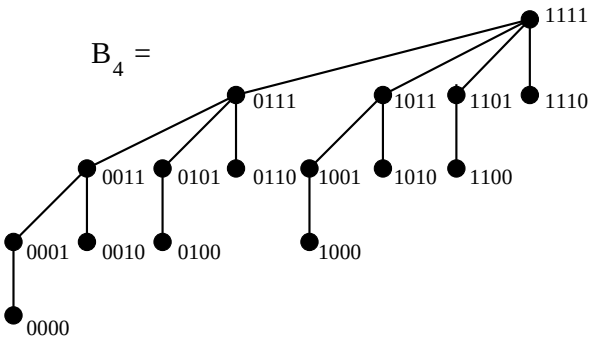
503-346-5373 (fax)

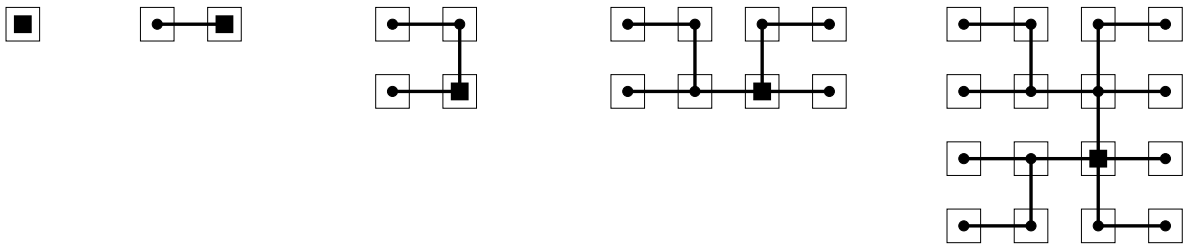
B_0 =

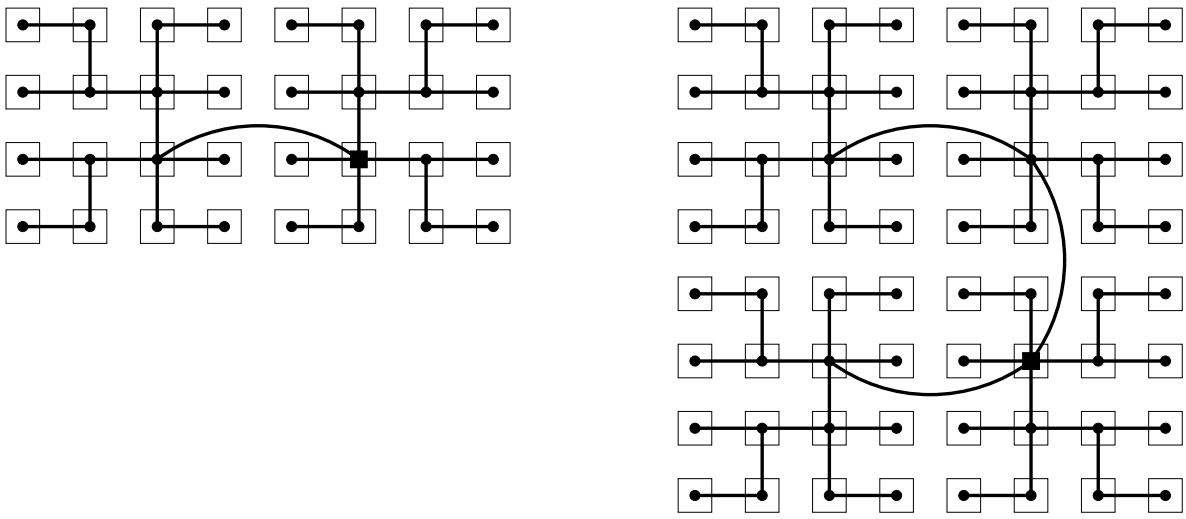
B_n =

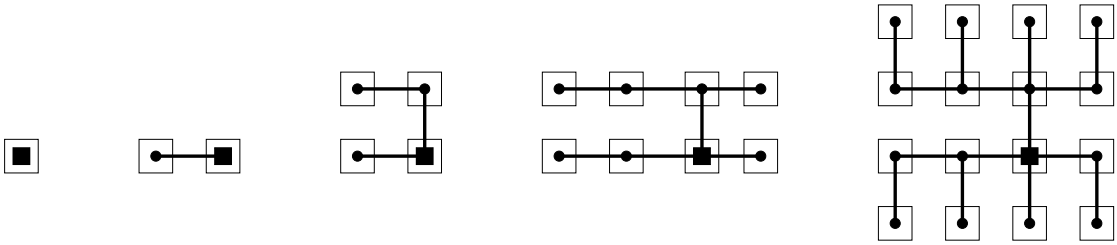

Bn-1
Bn-1

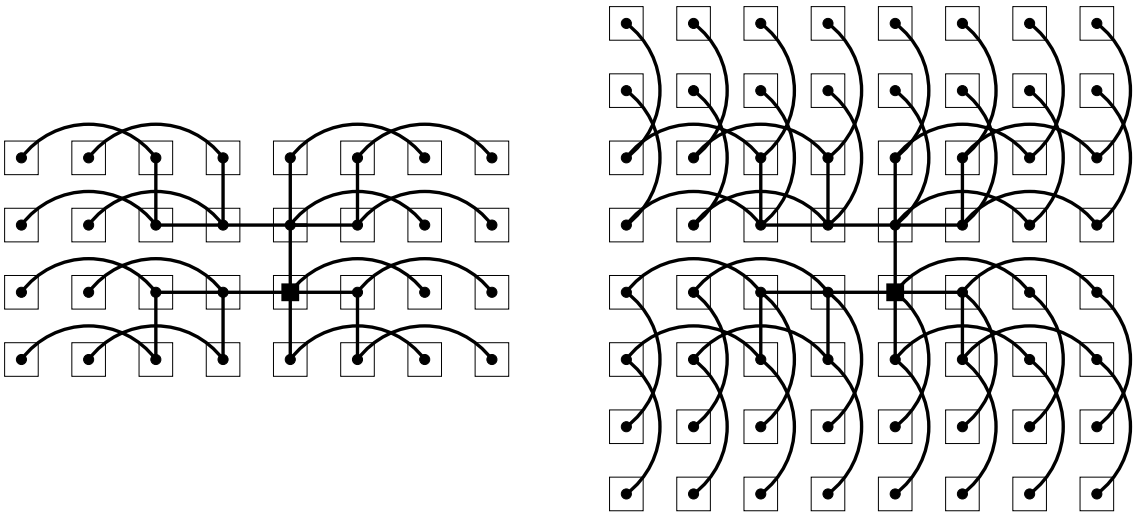
B_4 =

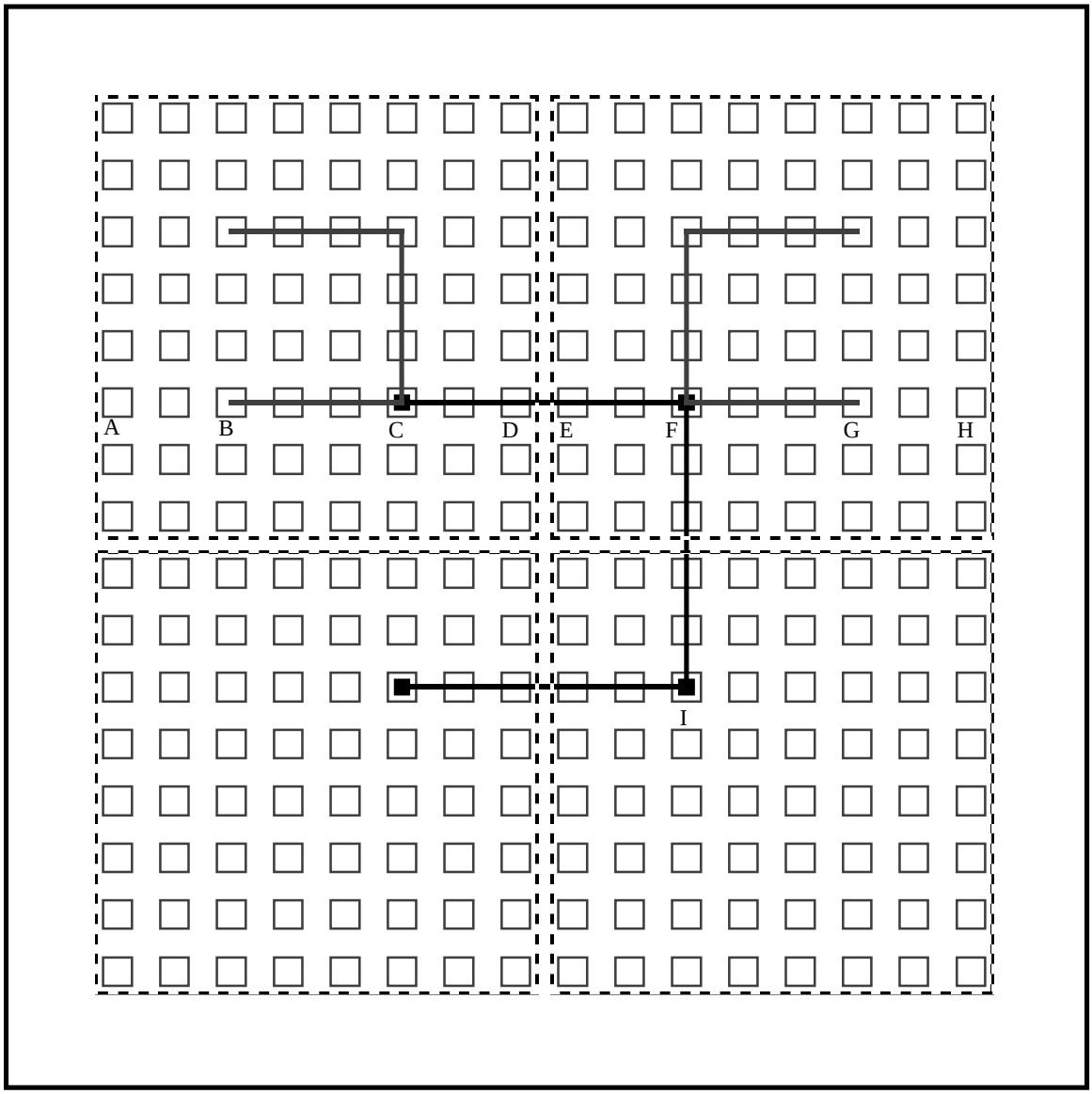

1111
0111
1011
1101
1110
0011
0101
0110
1001
1010
1100
0001
0010
0100
1000
0000










A
B
C
D
E
F
G
H
I