# Parallel Divide and Conquer on Meshes

Virginia Lo, Sanjay Rajopadhye,
Jan Telle and Xiaoxiong Zhong

CIS-TR-92-22
November 1992

Department of Computer and Information Science
University of Oregon

# Parallel Divide and Conquer on Meshes*

V. Lo, S. Rajopadhye†J. Telle, X. Zhong
Computer Science Department
University of Oregon
Eugene, OR 97403-1202

**Abstract**

This paper addresses 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 computations. We develop mappings which exploit the regular communication structure of this paradigm, including both the communication topology and the *phases* of message passing over time.

We develop several new performance metrics to evaluate the communication overhead of a *mapped* divide-and-conquer computation. These metrics are natural extensions of the standard contention and dilation metrics used in the literature, with refinements that take into account the underlying flow control scheme (wormhole vs. store and forward).

We present two mappings of the binomial tree divide-and-conquer computation to the 2-D mesh and evaluate their performance using our new metrics. Our first mapping, the Reflecting Mapping has optimal communication slowdown on a target machine with wormhole routing, independent of the message volumes. We also present a second mapping called the Growing Mapping which outperforms the Reflecting Mapping for a wide range of message volumes on a target machine with store and forward routing. We conclude that significant performance gains can be realized when the mapping of the divide and conquer computation to the 2-D mesh 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

†Present address: Electrical and Computer Engineering Department, Oregon State University, Corvallis OR 97330

# 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. Cole has even proposed fixed-degree divide and conquer as an *algorithmic skeleton* [Col89], and it has been studied by many researchers [AS88,Ble90,MH88,NS87]. Well known examples include mergesort, finding the max, parallel voting and broadcasting, applying an associative and commutative operation to a set of data values (such as multiplication of a sequence of matrices), parallel prefix computations, and many others. In this paper, we address the problem of implementing degree-two divide and conquer on two-dimensional meshes. We analyze our implementations using a single unified cost function that is designed to accurately measure the true performance. This work is a part of our ongoing effort on mapping parallel computations to distributed memory multiprocessors [LRM+90,LRG+91]. One of the goals in our research is to exploit regularity in the computation—rather than mapping a single task graph, we would like to develop a (parameterized) mapping function that maps an entire family of graphs.

The task structure of a divide and conquer algorithm is usually viewed as a complete binary tree (CBT), and it is widely assumed that it can be mapped to multiprocesors by embedding a CBT in the target topology. However, this is not accurate because of a number of reasons. First, a binomial tree is a better task graph than the CBT, because it has better speedup and efficiency [LRG+90]. Second, the traditional approach of graph embedding (using the conventional dilation and contention metrics [Ros88a,Ull84]) may not yield the best *performance* because of a number of pragmatic reasons. For one, this approach tacitly assumes that all the edges of the guest graph are simultaneously active. Second, it also ignores message volume (edge weights). Furthermore, many parallel computations, including divide and conquer, run in a step-by-step manner—there is a regular pattern in the times at wich messages are sent. Finally, the message volumes also exhibit considerable regularity. In this paper we develop parallel implementations that make effective use of all these facts.

We shall first present new cost functions for the mapping problem. They are developed specifically for phased comutations—parallel programs that run in a step by step manner. The metrics also address the facts that message volumes may not always be constant, and that communication technology can strongly affect the performance. 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. For each of these cases we develop a single, unified metric that represents the slowdown of the program relative to an ideal embedding. Our functions are extensions of the standard contention and dilation metrics used in the embedding literature.

Second, we study the structure of divide and conquer algorithms, specifically to exploit

regularity. We will justify why the binomial tree rather than the complete binary tree is a better task graph for these algorithms. We show that there is both *topological*, as well as *temporal* regularity in these algorithms. In addition, the patterns of message *volume* are also regular. This regularity can be described by two parameters $n$ and $\alpha$, where, $n$ simply represents the size of the graph, and $\alpha$ denotes the message size in each phase, as a fraction of the incoming message. Two common patterns, namely *uniform* communication (i.e., all messages have the same volume) and *logarithmic* communication (the message volume decreases by a factor of 2 at each level), merely correspond to specific values of $\alpha$. All relevant information needed for our mapping is completely described by these two parameters.

We then exploit these properties to implement divide and conquer on 2D mesh connected maxhines, using two different mapping functions. We analyze their performance using the metrics that we have developed, for the four cases based on routing scheme and message volumes. For each case, we identify the range of parameters of the divide and conquer where one implementation is better than the other. The remainder of the paper follows the above outline: Sec. 2 describes the regular structure of divide and conquer algorithms; Sec. 3 presents our cost functions; Sec. 4 presents our mappings and an analysis of their performance. Finally, we conclude with a discussion and indication of future work.

# 2 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). We call this the Type 1 strategy. The parameter $\alpha$, used in step 2 below, 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. Divide the problem into two equal parts, (each of size $\alpha x$), and send one part to each child (or, if the node is a leaf, solve the problem locally and skip the next step).
3. Get the results from the children and combine them.
4. Send the results to the parent. (the host, if the node is the root).

Many researchers have reported on parallel divide and conquer by studying the problem of mapping this graph structure to multiprocessor topologies [Col89,HZ83,MS87,Pet83]. However, this is not efficient since all non-leaf nodes (and hence about half the processes) are idle between steps 2 and 3. The computation actually 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. We call this a **phased** computation.

As is well known in the folklore and noted by many authors [AS88,Col89], an obvious way to improve the performance is to use the "keep half, send half" strategy. We call this the Type II strategy. The only change is to step 2, which now becomes:

2. Solve the problem locally (if 'small enough'), or divide the problem into two parts (each of size $\alpha x$), and spawn a child process *to solve one of them.* In parallel, start solving the other one, by repeating Step 2.

As before, there are two stages—*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. Because we are using divide and conquer as a general paradigm, some algorithms (such as prefix computations) may employ only one of the stages, while others (say, sorting) may use both. Nevertheless, 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.

The parameter $\alpha$ described above can serve to completely define the communication volumes that occur in the divide and conquer paradigm (we assume that $0 < \alpha \leq 1$). All edge weights can be expressed in terms of only $\alpha$ and the phase, $i$. Moreover, $\alpha$ 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, $\alpha$ is $\frac{1}{2}$ in the divide mode, and 1 in the combine mode. In such cases, the mapping that yields better performance for the dominant stage should be chosen.

It will soon be clear that the graph corresponding to the Type II strategy is a binomial tree. It has only about half the nodes as the corresponding CBT, but this does not affect the running time of the algorithm. Hence, if we can map the binomial tree efficiently on the target machine, we can save about half the processors without any time penalty. Alternatively, we can obtain a two fold speedup (for large enough input size) by executing an additional level of the algorithm in parallel (i.e., mapping the *next larger* binomial tree on the target machine).

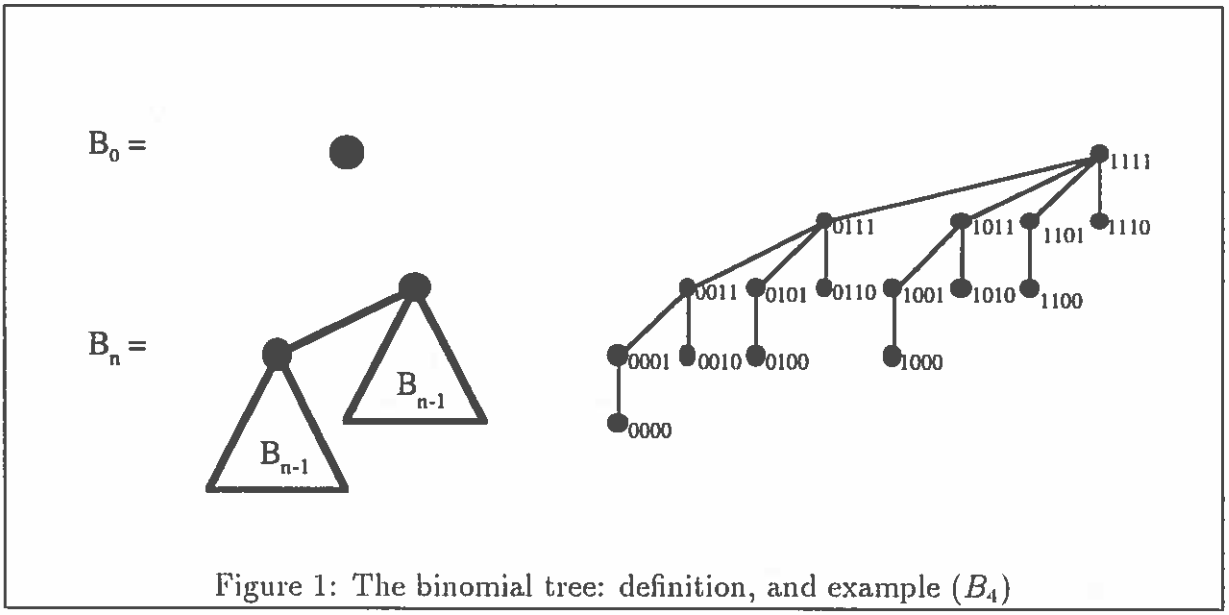


Figure 1: The binomial tree: definition, and example ($B_4$)

**Definition 1** The binomial tree $B(n)$ is defined inductively as follows [Vui87] (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 $B(n)$.

Because of the recursive definition, the subtree rooted at any node is itself a binomial tree. It is easy to see that 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 $r$ is the root of $B(n-i)$. We adopt a post order labelling of the tree as shown in Fig 1.

**Lemma 1** The computation graph of the Type II divide and conquer with $n$ divide phases is the binomial tree $B(n)$.

**Proof:** Let $C(n)$ be the, undirected, computation graph of the Type II divide and conquer with $n$ divide phases. $C(0)$ 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 $B(1)$ 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

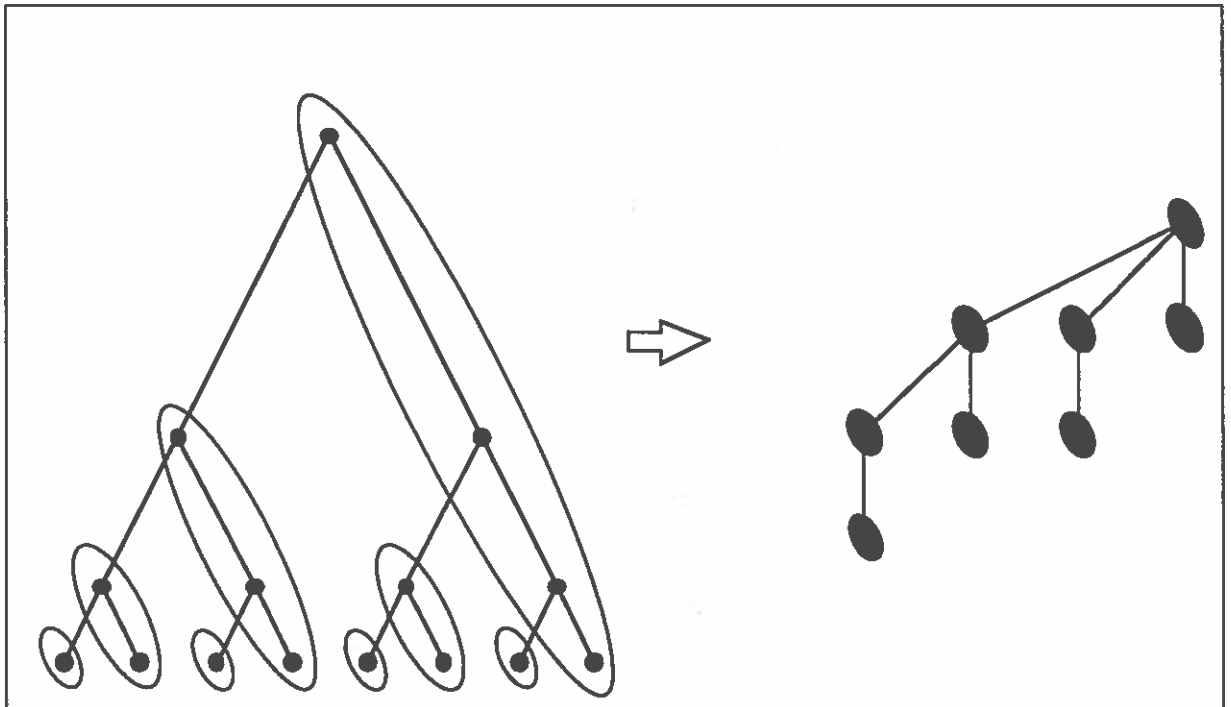

Figure 2: Binomial tree $B(3)$ as a contraction of the complete binary tree $CBT(4)$

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 $B(n+1)$ . ∎

From this proof it is clear that $B(n)$ has $2^n$ nodes compared to the $2^{n+1}-1$ nodes of the complete binary tree $CBT(n+1)$. Fig. 2 shows a contraction of $CBT(n+1)$ to $B(n)$ which compares Types I and II divide and conquer and illustrates the amount of work done by each task. For any node $u$ in $CBT(n+1)$, if $u$ is a right child of its parent $v$, then $u$ and $v$ are contracted into the same node. The contraction can be performed by starting at a leaf in $CBT(n+1)$ and tracing the parent relation towards the root until a node which is not a right child of its parent (or the root) is encountered. All nodes in this path are contracted into one.

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 $n$ (see Fig. 3). 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

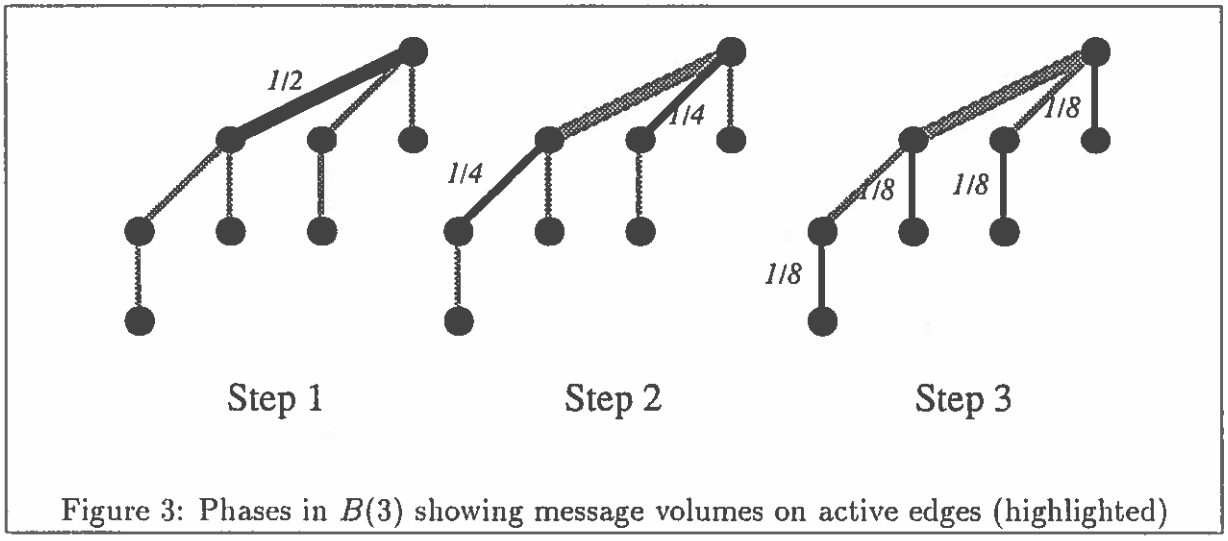


Figure 3: Phases in $B(3)$ showing message volumes on active edges (highlighted)

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 $\alpha$ 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 $\alpha^i$ to a child.

In the next section we develop metrics for estimating the total communication overhead when such phased computations are mapped to a multiprocessor.

# 3 New metrics for the mapping problem

The mapping problem typically uses the well known static task graph model of Stone [Sto77] and Bokhari [Bok87]. The parallel computation is viewed as a weighted graph[1], $G_C = \langle V_C, E_C, W \rangle$. 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 $\mathcal{M}_e$

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

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.

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,

[1] In general, nodes are also weighted; if the weights are all equal the computation is said to be uniform.

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 have elsewhere presented a language, to describe such graphs in a parameterized manner [LRM+90], and Lo [Lo92] has described the relationship between these graphs, the static task graph model, Lamport's processor time graphs [Lam78], and multiprocessor scheduling of DAGs [Pol88].

For the purposes of this paper, we assume that $|V_C| = |V_A|$, and $\mathcal{M}_n$ is one-to-one (this was justified in Sec 2). We measure the performance of a mapping by the communication overhead. In practice, this is a complex function of message volumes (edge weights), routing schemes and physical properties of the hardware (bandwidth, startup time, overhead for headers, etc.), and also the time instants at which communication occurs in the program. Hence, we need to use approximate metrics. We shall now develop a single cost function, communication slowdown, $\mathcal{S}$, to estimate this communication time for two common routing strategies—store-and-forward and wormhole, under two common ranges of message volumes. Our metric is based directly on the classic metrics used in graph embedding, namely contention and dilation, defined as follows [Ros88a,Ull84].

**Definition 2** 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 3** 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. Note that both dilation and contention are minimized if $G_C$ is a subgraph of $G_A$. This gives us a lower bound on the optimal cost of any mapping. Such a mapping, $\mathcal{P}$, is said to be **perfect**.

Traditionally, these two metrics have been used to compare multiprocessor networks by measuring the time taken for the host machine $G_A$ to simulate one step of an algorithm running on the guest machine $G_C$. In such analysis, one assumes that edge weights of $G_C$ are uniform and that all edges of $G_C$ are active simultaneously. These assumptions are reasonable in the context of comparing multiprocessor networks, but not sufficiently accurate for mapping. Below, we refine the definitions of dilation and contention to more realistically reflect communication overhead by taking into account (1) the communication phase behavior of the parallel computation, and (2) non-uniform edge weights. Note that we define contention slightly differently—in terms of contention between edges in the

computation graph, rather than links in the architecture. The reason for this will become clear shortly. All metrics are defined with respect to a specific mapping $\mathcal{M}$. In case of ambiguity, an additional subscript indicates the mapping. As a guide to the notation, we will define most terms as either functions of a single edge $e$ (which is active in the $i$-th phase), or of all edges in the $i$-th phase, $E_i$. The value of the function for a whole 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 4** The **i-th phase interference set**, $I(e,i)$ 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., $I(e,i) = \{e' \in E_i \mid e \neq e' \wedge \mathcal{M}_e(e) \cap \mathcal{M}_e(e') \neq \emptyset\}$. "Path level contention," as introduced by Chittor and Enbody [Chi91] is a similar notion, although they do not deal with phased computations.

**Definition 5** 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**, $\mathcal{C}_w(e,i)$, is the sum of the weights of edges in its i-th phase interference set, $\mathcal{C}_w(e,i) = \sum_{e' \in I(e,i)} W(e')$.
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, $\mathcal{C}_w(E_i) = \max_{e \in E_i} \mathcal{C}_w(e,i)$.

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, $\mathcal{C}_w(e,i) = |I(e,i)|$ and $\mathcal{C}_w(E_i) = \max_{e \in E_i} |I(e,i)|$. We indicate these **uniform** metrics by dropping the subscript $w$.

We now use Defn 5 to obtain estimates for the communication slowdown imposed by a mapping. For this, we first estimate the communication time, $\mathcal{T}(e,i)$ for an edge, $e \in E_i$, accounting for delays due to the path length and congestion. Then, 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., $\mathcal{T}(E_i) = \max_{e \in E_i} \mathcal{T}(e,i)$. The total communication time for the program, $\mathcal{T}(E_C)$ is the sum over all the phases, i.e., $\mathcal{T}(E_C) = \sum_i \mathcal{T}(E_i) = \sum_i \max_{e \in E_i} \mathcal{T}(e,i)$. Finally, we normalize this quantity with respect to the perfect mapping as follows.

**Definition 6** 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., $\mathcal{S}(\mathcal{M}) = \frac{\mathcal{T}_{\mathcal{M}}(E_C)}{\mathcal{T}_{\mathcal{P}}(E_C)}$

We will now show how the metrics of Defn 5 can be used to obtain formulae for $\mathcal{S}$ under two routing technologies, namely store and forward routing, and wormhole routing.

## 3.1 Store and Forward Routing (Large Message Volume)

Let us first consider a single edge, $e \in E_C$, in isolation. Empirical studies [SB90b] have indicated that in a multiprocessor with store-and-forward routing, the time taken to send a message[2], $e$, of volume $W(e)$ from one processor to another over a path of length $d$ is $t(e) = (c + bW(e))d$, where $c$ is a startup cost (overhead of creating and buffering a message), and $1/b$ is the bandwidth of the link. Note that startup overhead is incurred once for each link in the path, since each processor in the path is required to receive the message entirely, and then forward it. It also well known that $c$ is about three orders of magnitude larger than $1/b$. Indeed, a well known rule of thumb is that the message size must be about 1k-bytes to offset the startup overhead. Also note that any link $l$, is busy with a message for exactly $c + bW(e)$ time units; once the message has passed, the link can be released. For the present, we assume (this is relaxed later on) that $c \ll bW(e)$, i.e., the message volumess are large enough that startup costs can be safely neglected. Thus, $t(e) = bW(e)d$. This assumption is relaxed later, when we consider small message volumes.

Let us now consider what happens when there is contention. In order to estimate the total communication time, we assume that the message $e$ will meet and be delayed by *every* message $e'$, in its i-th phase interference set, $I(e,i)$. We further assume that the delay induced by each $e'$ is equal to the time that $e'$ needs to occupy a single link.

The first assumption is pessimistic, since there may not necessarily be a conflict merely because two messages happen to use the same link (the first message may arrive early enough that by the time the second message arrives, the link may be free). To understand the justification for the second assumption, it is obviously true if the messages contend exactly once for a single link. One of them is then delayed by precisely the time that the other one occupies a link. If two messages contend for a contiguous sequence of links, then they are sent in a pipelined fashion, one after the other. In this case, the delay for the first link is just what we have assumed above. For each successive link, the delay depends on the difference between the message volumes (if the message that is sent first is no larger than the later one, then there is no further slowdown). Finally, if two messages contend multiple times in disjoint sections of their paths our assumption is not valid , although it is often the case that the first conflict will separate the messages enough that later contention is unlikely in any but the most pathological cases. Our assumption thus yields a reasonable approximation. $\mathcal{T}(e,i)$ is then given by the sum of two terms, one accounting for the "natural" time for $e$ and one for message congestion among the $i$-th phase contention set, as follows.

$$\begin{aligned}\mathcal{T}(e,i) &= bW(e)\mathcal{D}(e) + \sum_{e' \in I(e,i)} bW(e') \\ &= b[\mathcal{D}_w(e) + \mathcal{C}_w(e,i)]\end{aligned}$$

[2]We will abuse the notation and consider an edge to be synonymous to the message that it carries.

Thus, the expected communication time for the i-th phase is as follows (note that the final expression is an upper bound which we will use as an approximation):

$$\begin{aligned}
\mathcal{T}(E_i) &= \max_{e \in E_i} \mathcal{T}(e,i) \\
&= b \max_{e \in E_i} [\mathcal{D}_w(e) + \mathcal{C}_w(e,i)] \\
&\leq b[\max_{e \in E_i} \mathcal{D}_w(e) + \max_{e \in E_i} \mathcal{C}_w(e,i)] \\
&= b[\mathcal{D}_w(E_i) + \mathcal{C}_w(E_i)]
\end{aligned}$$

The total communication time for the entire algorithm with $k$ phases (under the mapping $\mathcal{M}$) is then obtained by summing the communication time over all the phases:

$$\mathcal{T}_{\mathcal{M}}(E_C) = \sum_{i=1}^{k} \mathcal{T}(E_i) = b \sum_{i=1}^{k} [\mathcal{D}_w(E_i) + \mathcal{C}_w(E_i)]$$

To normalize this, recall that under the perfect mapping, the communication time for any edge, $e$ is just $W(e)$ (since dilation is 1 and there is no contention). Thus, $\mathcal{T}_{\mathcal{P}}(E_i) = \max_{e \in E_i} bW(e)$, and

$$\mathcal{T}_{\mathcal{P}}(E_C) = b \sum_{i=1}^{k} W(E_i) \tag{1}$$

Hence the single performance measure for store-and-forward routing is given by the following.

**Remark 1** The **slowdown**, $\mathcal{S}(\mathcal{M})$, of a mapping, $\mathcal{M}$ to a machine with store-and-forward routing (for large message volumes) is given by

$$\boxed{\mathcal{S}(\mathcal{M}) = \frac{\sum_{i=1}^{k} [\mathcal{D}_w(E_i) + \mathcal{C}_w(E_i)]}{\sum_{i=1}^{k} W(E_i)}} \tag{2}$$

In Eqn 2, note that the denominator is independent of the mapping, and is an inherent characteristic of the program. The communication slowdown is thus determined primarily by the sum (for each phase) of the sum of the $i$-th phase weighted contention and dilation of the mapping.

## 3.2 Wormhole Routing (Large Message Volume)

In wormhole routing, a message consists of a stream of *contiguous* data (called flits), routed through the network with minimal buffering. Thus the overhead of each intermediate

node having to buffer the complete message is avoided, but if a message is blocked due to conflict for a communication link, it stays in the network (which may in turn, block other messages) until the link is released. To develop the communication time estimates, let us again consider a single edge, $e \in E_C$, in isolation. Because of pipelining, the time it takes to travel from one processor to another over a path of length $d$ is $t(e) = c + b(W(e) + dh)$, where, $c$ is the startup cost, $1/b$ is the bandwidth, and $h$ is the header size (usually, 1 or 2 bytes) [SB90b]. It is known that $c \gg bdh$, so $t(e) = c + bW(e)$. Note that the distance is no longer relevant; this has been observed by many authors [Bok90,Dun91]. For large volumes, $t(e) = bW(e)$,

When two messages contend for the same link, one of them is delayed until the other has been *completely* transmitted. Once again, we assume that an edge $e$ is delayed by *all* edges, $e' \in I(e,i)$, and that the delay caused by each $e'$ is equal to its own natural communication time. As before, there are many factors that affect this value, such as the specific instant when the messages meet, whether any of the other messages are themselves blocked, etc., but this is a reasonable approximation. Thus, from Defn 5

$$\mathcal{T}(e,i) = b[W(e) + \mathcal{C}_w(e,i)]$$

Then we have the following:

$$\begin{aligned} \mathcal{T}(E_i) &= \max_{e \in E_i} \mathcal{T}(e,i) \\ &= b \max_{e \in E_i} [W(e) + \mathcal{C}_w(e,i)] \\ &\leq b[W(E_i) + \mathcal{C}_w(E_i)] \end{aligned} \tag{3}$$

and the communication time for the entire paralle program with $k$ phases under the mapping $\mathcal{M}$ is

$$\mathcal{T}_{\mathcal{M}}(E_C) = \sum_{i=1}^{k} \mathcal{T}(E_i) = b \sum_{i=1}^{k} [W(E_i) + \mathcal{C}_w(E_i)]$$

Using Eqn 1, we get,

**Remark 2** The **slowdown**, $\mathcal{S}(\mathcal{M})$, of a mapping, $\mathcal{M}$ to a machine with wormhole routing (for large message volumes) is given by

$$\boxed{\mathcal{S}(\mathcal{M}) = 1 + \frac{\sum_{i=1}^{k} \mathcal{C}_w(E_i)}{\sum_{i=1}^{k} W(E_i)}} \tag{4}$$

In Eqn 4, the denominator is again independent of the mapping. Also note that dilation does not affect the performance. The slowdown differs from the optimal value by the phasewise sum of the $i$-th phase weighted contention (normalized by the communication time of the perfect mapping).

### 3.3 Small Message Volumes

In the preceeding development, we assumed that the startup costs of message transmission was negligible. We shall now consider the other extreme, i.e., when startup costs dominate. Recall that the "natural" time $t(e)$, for a single edge, is $(c + bW(e))d$ with store-and-forward routing, and $c + bW(e)$ with wormhole routing. This reduces to $cd$ and $c$, respectively, if $c \gg bW(e)$, and thus, the analysis is identical to the large volume case with **uniform** edge weights, $W(e) = c/b$.

For store and forward routing (note that we simply use the uniform metrics, and have a different constant factor),

$$\mathcal{T}_{\mathcal{M}}(E_C) = c\sum_{i=1}^{k}[\mathcal{D}(E_i) + \mathcal{C}(E_i)]$$

and for wormhole routing (the inequality in Eqn 3 now becomes an equality, as the reader may easily verify),

$$\mathcal{T}_{\mathcal{M}}(E_C) = c[k + \sum_{i=1}^{k}\mathcal{C}(E_i)]$$

By a reasoning similar to the large volume case, we can show that,

$$\mathcal{T}_{\mathcal{P}}(E_C) = b\sum_{i=1}^{k} W(E_i) = ck \tag{5}$$

Hence, we have,

**Remark 3** For small message volumes the slowdown, $\mathcal{S}(\mathcal{M})$, of a mapping, $\mathcal{M}$ to a machine with store and forward routing is given by

$$\boxed{\mathcal{S}(\mathcal{M}) = \frac{1}{k}\sum_{i=1}^{k}[\mathcal{D}(E_i) + \mathcal{C}(E_i)]} \tag{6}$$

For wormhole routing, it is given by

$$\boxed{\mathcal{S}(\mathcal{M}) = 1 + \frac{1}{k}\sum_{i=1}^{k}\mathcal{C}(E_i)} \tag{7}$$

# 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). The first mapping uses Definition 3 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. Both mappings have load 1 and expansion 1. In both mappings, adjacent nodes of the binomial tree are mapped to processors in either the same column or the same row of the mesh. Edges are mapped to the unique shortest path connecting their endpoints. Thus, the mappings can be completely specified by the node mapping.

We analyze both mappings with respect to the metrics presented earlier for a spectrum of message-passing volumes as represented by the parameter $\alpha$. We show that the Reflecting Mapping has optimal slowdown for a target machine with wormhole routing, for all values of $\alpha$. However, its performance for store-and-forward routing is poor for computations with logarithmically decreasing message volume ($\alpha = 1/2$). We show that the Growing Mapping outperforms the Reflecting Mapping for a target machine with store-and-forward routing and analyze its behavior for a spectrum of values of $\alpha$.

**Note:** In the analysis presented in the following sections, the routing parameter $b$ (*bandwidth*) is a multiplicative factor in the communication overhead metrics and ultimately cancels out. Thus, for ease of exposition, we assume for the remainder of the paper that it is unity, $b = 1$.

## 4.1 The Reflecting Mapping

**Definition 7** The reflecting mapping $M_R$ is defined inductively as follows:(see Fig 4).

- $B(0)$, 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 $B(n)$.

Because of the successive reflection above and the post-order labeling used for $B(n)$, the processor label $[x, y]$ to which a node $b = b_{n-1} \ldots b_1 b_0$ in the binomial tree is mapped, is related to the binary reflected gray code. Indeed, because of the alternation in how the

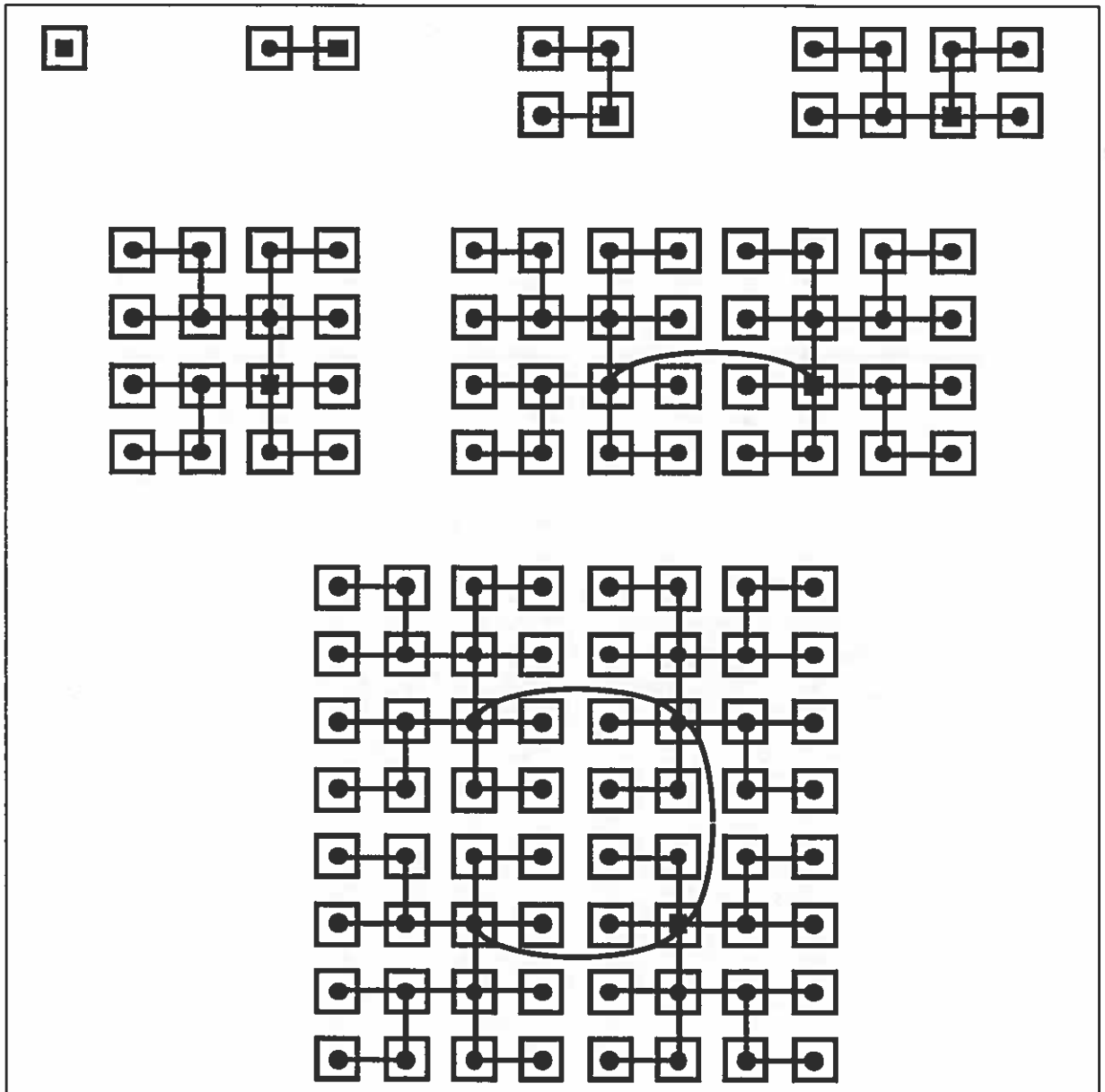

Figure 4: The reflecting mapping $M_R$ (the square node is the root)

reflection is performed, alternate bits of $b$ determine the $x$ and $y$ co-ordinates respectively, and it can be shown by induction that the processor label $[x, y]$ to which a node $b = b_{n-1} \ldots b_1 b_0$ of $B(n)$ is mapped is given by

$$[x, y] = M_R(b_{n-1} \ldots b_1 b_0) = [\text{gray}(\ldots b_2 b_0), \text{gray}(\ldots b_3 b_1)]$$

where $\text{gray}(\ldots b_2 b_0)$ and $\text{gray}(\ldots b_3 b_1)$ are the gray codes corresponding to the even and odd indices, respectively, of the binary number $b_{n-1} \ldots b_1 b_0$.

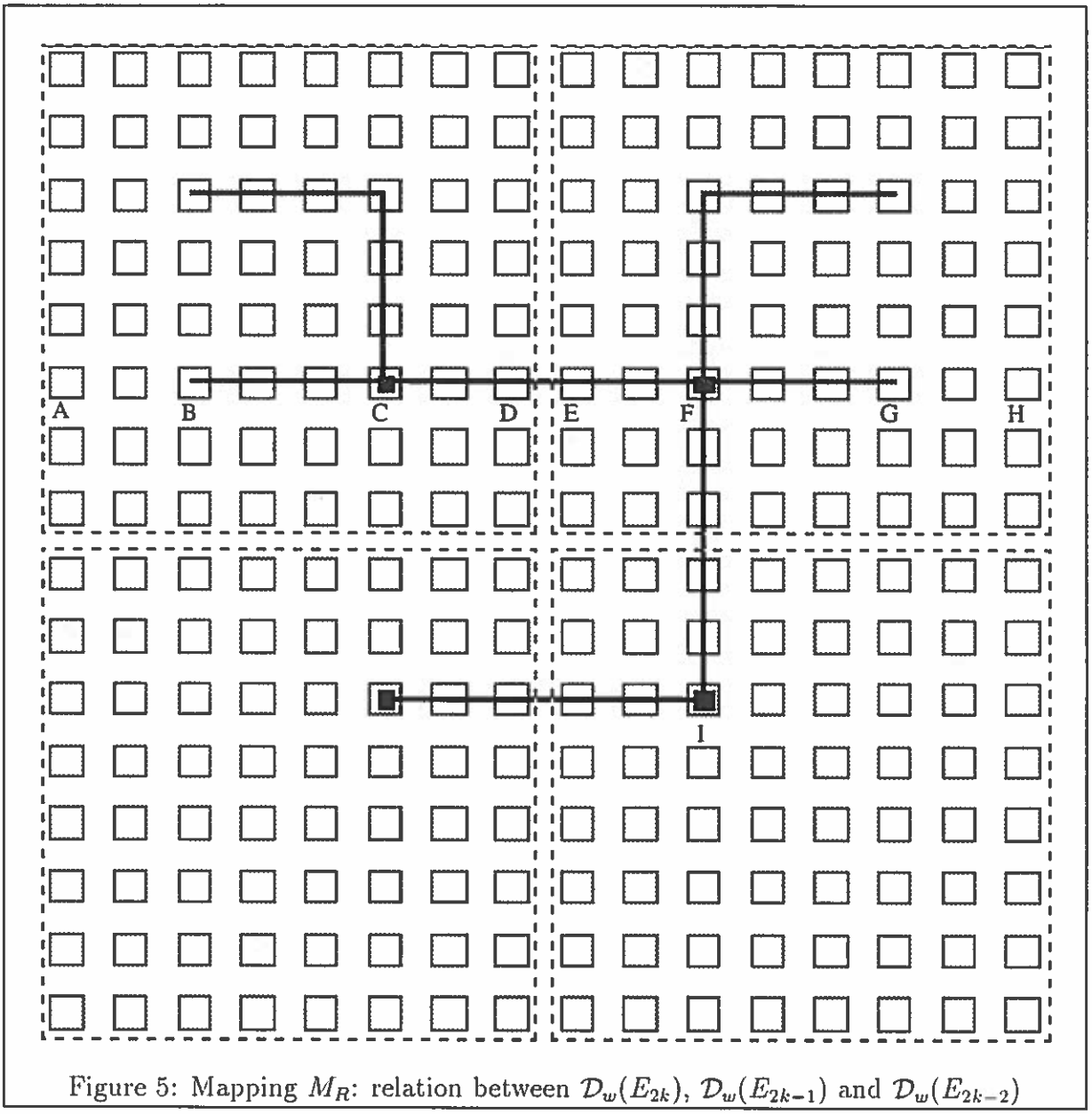


Figure 5: Mapping $M_R$: relation between $\mathcal{D}_w(E_{2k})$, $\mathcal{D}_w(E_{2k-1})$ and $\mathcal{D}_w(E_{2k-2})$

To analyze the performance of the reflecting mapping of $B(2k)$ we next establish the maximum weighted contention $\mathcal{C}_w(E_i)$ and dilation $\mathcal{D}_w(E_i)$ for each phase.

**Lemma 2** For 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 $1 \leq i \leq n$.

**Proof:** We prove this by induction on $n$. $B(0)$ 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 $B(n)$. ∎

**Lemma 3** The maximum weighted dilation for the reflecting mapping in phase $i$ is given by

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

**Proof:** We have already argued that weights of all edges in phase $i$ is $\alpha^i$. 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 5 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

$$\mathcal{D}(E_{2k}) \quad = 2^{k-1} - \mathcal{D}(E_{2k-2}) \tag{8}$$

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. The solution to the recurrence is found in the appendix and when multiplied by $\alpha^i$ establishes the Lemma for even $i$. For odd $i$, we have already argued that $\mathcal{D}(E_{2k-1}) = \mathcal{D}(E_{2k})$. ∎

We now state a theorem giving the slowdown of the reflecting mapping for a target machine with wormhole routing. From equations (4) and (7) it is clear that a mapping which has no contention in any phase has optimal slowdown, so this result is an easy corollary of lemma 2.

**Theorem 1** The reflecting mapping has optimal slowdown on a target machine with wormhole routing, for both small and large message volumes and for any value of $\alpha$, $S(M_R) = 1$.

For machines with store-and-forward routing, we will want to compare the performance of $M_R$ with that of the growing mapping $M_G$ described in the next section. From equations

(2) and (6) we note that it suffices to evaluate the total communication time $\mathcal{T}_{\mathcal{M}}(E_C) = \sum_{i=1}^{2k}[\mathcal{D}_w(E_i) + \mathcal{C}_w(E_i)]$ (as stated before, $b = 1$) for both mappings to compare their respective slowdowns. Note that comparison for small message volumes, equation (6), corresponds to comparison of $\mathcal{T}_{\mathcal{M}}(E_C)$ at $\alpha = 1$. From lemmata 2 and 3, we get

$$\mathcal{T}_{M_R}(E_C) = \sum_{i=1}^{2k}[\mathcal{D}_w(E_i) + \mathcal{C}_w(E_i)] = \sum_{i=1}^{2k}[(\frac{2^{\lceil \frac{i}{2} \rceil} - (-1)^{\lceil \frac{i}{2} \rceil}}{3})\alpha^i + 0]$$

whose solution for the 3 cases, general $\alpha$, $\alpha = 1$ and $\alpha = \frac{1}{2}$, is summarized in the following lemma.

**Lemma 4** 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 $\alpha = \frac{1}{2}$

## 4.2 The Growing Mapping

For store-and-forward routing and $0 < \alpha < 1$, the reflecting mapping is not optimal, indeed the volume of a message is proportional to its dilation. For this reason, we present the growing mapping $M_G$ for which the volume of a message is inversely proportional to its dilation.

**Definition 8** Recall the definition of $B(n)$ arising from Lemma 1, which creates $B(n)$ by growing a leaf from each node of a copy of $B(n-1)$. The map-node function of $M_G$ uses this definition and is specified inductively below (see figure 6). Edges are mapped to the unique shortest mesh-path between their endpoints.

- For $B(0), B(1), B(2)$ the map-node function is the same as for $M_R$.
- 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).

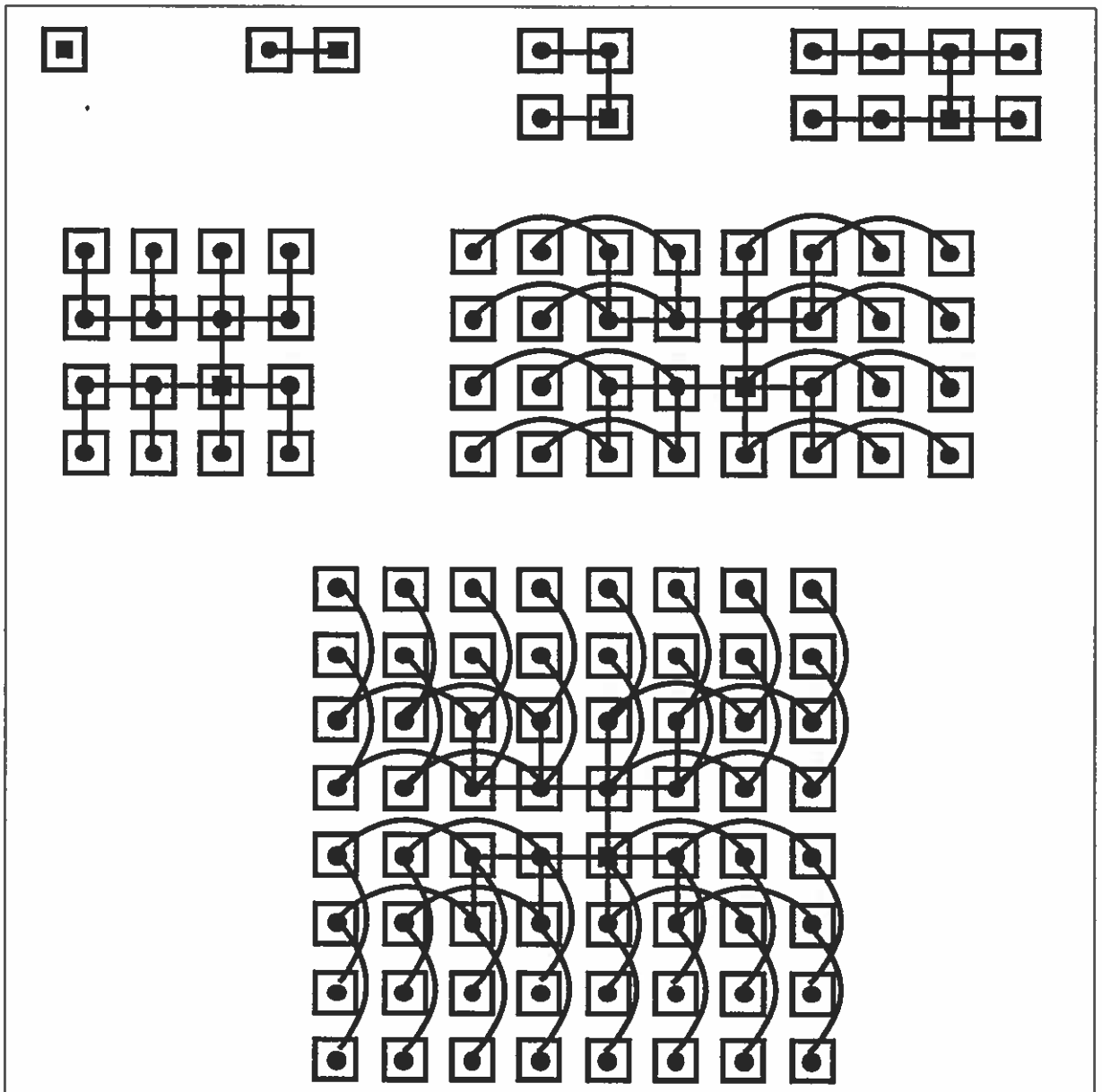

Figure 6: The growing mapping $M_G$ (the square node is the root)

- 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).

In phases 1, 2, 3 and 4 $M_G$ has dilation 1 and no contention. In general, for phase $i > 2$, we have the volume of each edge $\alpha^i$, the dilation of each edge $2^{\lceil \frac{i}{2} \rceil - 2}$ and the size of the conflict set of each edge $2^{\lceil \frac{i}{2} \rceil - 2} - 1$, so that $\mathcal{D}_w(E_i) = \alpha^i(2^{\lceil \frac{i}{2} \rceil - 2})$ and $\mathcal{C}_w(E_i) = \alpha^i(2^{\lceil \frac{i}{2} \rceil - 2} - 1)$. As an aside we mention that the size of the conflict sets can be halved in

each phase by growing edges horizontally and vertically for alternating vertices in each row and column. However, this complicates the description of the mapping considerably and also results in a non-rectangular image of $B(2k+1)$.

Since the reflecting mapping is optimal with respect to wormhole routing, we consider the growing mapping only with respect to store-and-forward, for which we get the total communication time for $B(2k)$

$$\mathcal{T}_{M_G}(E_C) = \sum_{i=1}^{2k}[\mathcal{D}_w(E_i) + \mathcal{C}_w(E_i)] = \alpha + \alpha^2 + \sum_{i=3}^{2k}[\alpha^i(2^{\lceil \frac{i}{2} \rceil - 1} - 1)]$$

whose solution for the cases, general $\alpha$, $\alpha = 1$, $\alpha = \frac{1}{2}$ and the special case $\alpha = \frac{1}{\sqrt{2}}$, is summarized in the following lemma.

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

(a). $\mathcal{T}_{M_G}(E_C) = c_5(2\alpha^2)^k + c_6 + c_7\alpha^{2k}$ for $0 < \alpha < 1$ but $\alpha \neq \frac{1}{\sqrt{2}}$

where $c_5 = \frac{\alpha(1+\alpha)}{2\alpha^2-1}$ , $c_6 = \frac{2\alpha^3(1+\alpha)}{1-2\alpha^2} + \frac{1}{\alpha-1} + 1 + 2\alpha + 2\alpha^2$ , $c_7 = \frac{\alpha}{1-\alpha}$

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

(c). $\mathcal{T}_{M_G}(E_C) = 1.25 - (\frac{3}{2^{k+1}} - \frac{1}{2^{2k}})$ for $\alpha = \frac{1}{2}$

(d). $\mathcal{T}_{M_G}(E_C) = \Theta(k)$ for $\alpha = \frac{1}{\sqrt{2}}$.

Note that the denominators of $\mathcal{T}_{M_G}(E_C)$ are 0 for $\alpha = \frac{1}{\sqrt{2}}$ and $\alpha = 1$.

## 4.3 Comparisons of the two mappings

For machines with wormhole routing the reflecting mapping is optimal (Theorem 1) whereas the growing mapping clearly has some contention for each phase $i > 4$. In the following we compare our two mappings for machines with store-and-forward routing only. As mentioned previously, it will suffice to compare the total communication volume for each of the two mappings. We will also calculate the exact slowdown for the common cases $\alpha = 1$ and $\alpha = \frac{1}{2}$. We will need the following easy lemma:

**Lemma 6** The total communication time of the perfect mapping of $B(n)$ is given by

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

Comparisons between the two mappings are summarized in the next theorem.

**Theorem 2** For machines with store-and-forward routing our two mappings have the following metrics.

- For the large volume case with $\alpha = 1$ and also for the small volume case with arbitrary values of $\alpha$, the reflecting mapping outperforms the growing mapping by a factor of $\frac{2}{3}$. For $N$ processors they both have slowdown $\mathcal{S}(M_R) = \mathcal{S}(M_G) = \Theta(\frac{\sqrt{N}}{\log N})$.
- For the large volume case with any $0 < \alpha < 1$ there exists $k$ so that the growing mapping outperforms the reflecting mapping for $B(2k)$.
- For the large volume case with $\alpha = \frac{1}{2}$ the growing mapping has slowdown $\mathcal{S}(M_G) = \mathcal{O}(1)$, asymptotically approaching 1.25, whereas the reflecting mapping has slowdown $\mathcal{S}(M_R) = \Theta(2^k)$.

**Proof:** For $\alpha = 1$, we see from Lemmata 4.b and 5.b that assymptotically, as $k$ increases, the reflecting mapping outperforms the growing mapping by a factor of $\frac{2}{3}$. From equation (2) we see that the slowdown for the reflecting mapping for uniform communication and large message volumes is achieved by dividing the result in Lemma 4.b with the total communication time of the perfect mapping. This latter value for $B(2k)$ is $2k$ as given by Lemma 6. Disregarding the additive constant, the slowdown for uniform communication is $\mathcal{S}(M_R) = \frac{\mathcal{T}_{M_R}(E_C)}{\mathcal{T}_{\mathcal{P}}(E_C)} = \frac{2^{k+2}}{6k} = \frac{4\sqrt{N}}{3\log N}$, where $N = 2^{2k}$ is the number of processors used. From equation (6) we see that this also establishes the slowdown for small message volumes, for any value of $\alpha$.

The next result follows by comparing lemmata 4.a and 5.a and noting that when $0 < \alpha < 1$ we can always find $k$ so that $(2\alpha^2)^k < 2^k$.

For $\alpha = \frac{1}{2}$, the perfect mapping has total communication time $\mathcal{T}(\mathcal{P}) = \sum_{i=1}^{2k} \frac{1}{2^i}$ asymptotically approaching 1 (Lemma 6). The slowdown given in the theorem follows from the bounds given by lemmata 4.c and 5.c. ∎

# 5 Related Work

The divide and conquer model has been widely recognized as an effective parallel programming paradigm [NS87,Col89]. 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 [MH88].

The "keep half, send half" strategy to improve the efficiency of a divide and conquer algorithm was observed in [Col89,AS88]. However, the model is not formalized and the underlying binomial tree structure was not recognized in those work. In [Joh90], a spanning binomial tree of a hypercube was used to achieve efficient broadcasting. In [Ull84],

an H-tree embedding was proposed to embed a complete binary tree to a mesh. The embedding requires a slightly larger mesh.

Graph embedding technique is usually used to emulate a machine of one topology with another of a different topology [Ros88a,Ull84]. The maximum dilation and congestion (contention) 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 [Sto77,Bok87], such an approach is usually used to assign tasks to processors [Bok87]. However, as we have argued in the paper, this 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 [Sea88,Dal90,Bok90].

Much work has been done to analytically model a network with different routing schemes [KK79,Dal90,Aga90]. However, these models are all based on some assumpstions on the message distribution and thus not accurate enough for a specific application. The metrics analysis in this paper complements with these studies and gives important metrics in developing a mapping onto a network with store-forward or wormhole communication technology.

# 6 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.

Our first algorithm, the Reflecting Mapping has optimal communication slowdown on a target machine with wormhole routing, independent of the message volumes. However, for store-and-forward routing this mapping exhibits linear growth as a function of the number of processes with respect to communication overhead for logarithmically decreasing message volumes. The second mapping algorithm, the Growing Mapping, has constant slowdown (asymptotically approaching 1.25) in this case. The Growing Mapping outperforms the Reflecting Mapping for a wide range of message volumes on a target machine with store and forward routing. The results are summarized in Theorems 1 and 2. This indicates that the performance of algorithms for mapping parallel computations to paral-

lel architectures is sensitive to communication topology, communication volume, temporal communication patterns, and the routing scheme used by the communication hardware of the parallel machine.

This research was conducted within the context of the OREGAMI project, whose goal is the design of abstractions and algorithms for mapping parallel computations to parallel architectures. The OREGAMI software tools include (1) LaRCS, a language for describing the (regular) spatial and temporal communication structure of the computation to be mapped, (2) MAPPER, a library of mapping algorithms, and (3) METRICS, an X-windows based tool for static performance anaylsis of mappings produced by MAPPER. Currently under development is a multicomputer simulator for dynamic analysis of mapped computations. The two mapping algorithms presented here are part of the "canned mappings" found in the MAPPER library.

The ongoing and future work most closely related to the work presented here includes

- Development and testing of "canned mappings." The OREGAMI library of canned mappings includes those we have developed and a selection of mappings from the graph embedding literature. We are currently evaluating the performance of these mappings/embeddings in the practical context of specific routing schemes and non-uniform message volumes. Our goal here is to determine the practical utility of mapping algorithms based on graph theoretic abstractions of the parallel computation and the utility of using temporal information to guide the mapping.

- Validation of the performance metrics proposed here through extensive simulation, and comparision with existing metrics [SB90a,KS90,Chi91].

# References


[Aga90] Anant Agarwal. Limits on network performance. MIT VLSI Memo, 1990.

[AS88] W. C. Athas and C. L. Seitz. Multicomputers: Message-passing concurrent computers. *IEEE Computer*, pages 9–23, August 1988.

[Ble90] Guy E. Blelloh. *Vector models for data-parallel computing.* The MIT Press, 1990.

[Bok87] S. H. Bokhari. *Assignment Problems in Parallel and Distributed Computing.* Kluwer Academic Publishers, 1987.

[Bok90] S.H. Bokhari. Communication overhead on the intel iPSC-860 hypercube. Technical Report, ICASE, NASA Langley Research Center, May 1990.

[Chi91] S.S. Chittor. *Communication Performance of Multicomputers.* PhD thesis, Dept. of Computer Science, Michigan State University, 1991.

[Col89] M.I. Cole. *Algorithmic Skeletons:Structures Management of Parallel Computation.* MIT Press, 1989. Research Monographs in Parallel and Distributed Computing.

[Dal90] W.J. Dally. Performance analysis of $k$-ary $n$-cube interconnection networks. *IEEE Trans. Comput.*, C-39(6):775–785, June 1990.

[Dun91] T.H. Dunigan. Performance of the Intel iPSC/86- and Ncube 6400 hypercubes. *Parallel Computing*, 17:1285–1302, 1991.

[HZS3] E. Horowits and A. Zorat. Divide and conquer for parallel processing. *IEEE Transactions on Computers*, TC-32(6):582–585, June 1983.

[Joh90] S. L. Johnsson. Communication in network architectures. In *VLSI and Parallel Computation*, page page 290. Morgan Kaufmann Publishers, Inc., 1990.

[KK79] P. Kermani and L. Kleinrock. Virtual cut-through: a new computer communication switching technique. *Computer Networks*, 3:267–286, 1979.

[KS90] D.D. Kandlur and K.G. Shin. Traffic routing for multi-computer networks with virtual cut-through capability,. In *Preceedings of the 10th International Conference on Distributed Computer Systems,*, pages 398–405, May 1990.

[Lam78] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communication of ACM*, 21(7), July 1978.

[Lo92] V. M. Lo. Temporal communication graphs: Lamport's process-time graphs augmented for the purpose of mapping and scheduling. *To appear Journal of Parallel and Distributed Computing, Special Issue on Mapping and Scheduling*, 14(4), December 1992.

[LRG+90] V. M. Lo, S. Rajopadhye, S. Gupta, D. Keldsen, M. A. Mohamed, and J. Telle. Mapping divide-and-conquer algorithms to parallel architectures. In *Proceedings IEEE 1990 International Conference on Parallel Processing*, pages III:128–135, August 1990.

[LRG+91] V. M. Lo, S. Rajopadhye, S. Gupta, D. Keldsen, M. A. Mohamed, J. Telle, and X. Zhong. OREGAMI: Tools for mapping parallel algorithms to parallel architectures. *International Journal of Parallel Programming*, 20(3), June 1991.

[LRM+90] V. M. Lo, S. Rajopadhye, M. A. Mohamed, S. Gupta, B. Nitzberg, J. A. Telle, and X. Zhong. LaRCS: A language for describing parallel computations for the purpose of mapping. Technical Report CIS-TR-90-16, University of Oregon Dept. of Computer Science, 1990. To appear IEEE Transaction on Parallel and Distributed Systems.

[MS87] D. L. McBurney and M. R. Sleep. Experiments with the ZAPP: Matrix multiply on 32 transputers, heuristic search on 12 transputers. Technical Report SYS-C87-10, University of East Anglia, School of Information Systems, 1987.

[MH88] 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.

[NS87] 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.

[Pet83] F. J. Peters. Tree machines and divide and conquer algorithms. In *Proceedings, CONPAR 81, LNCS 111*, pages 25–36. Springer Verlag, 1983.

[Pol88] C. D. Polychronopoulos. *Parallel Programming and Compilers.* Kluwer Academic Publishers, 1988.

[Ros88a] A. L. Rosenberg. Graph embeddings 1988: Recent breakthroughs new directions. Technical Report 88-28, University of Massachusetts at Amherst, March 1988.

[SB90a] B. Stramm and F. Berman. How good is good? Technical Report CS90-169, University of California at San Diego, 1990.

[SB90b] Robert Suaya and Graham Birtwistle. VLSI and Parallel Computation,Chapter 3. San Mateo, California, 1990. Morgan Kaufmann Publishers, Inc.

[Sea88] C.L. Seitz et al. The performance and programming of the Ametek series 2010 multicomputer. In *Hypercube Computer Conference*, pages 33–36, 1988.

[Sto77] H. S. Stone. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Transactions on Software Engineering*, SE-3(1):85–93, January 1977.

[Ull84] J..D. Ullman. *Computational aspects of VLSI.* Computer Scienece Press, Rockville,MD, 1984.

[Vui87] J. Vuillemin. A data structure for manipulating priority queues. *Communications of the ACM*, 21(4):309–315, April 1987.