Communication Complexity for Parallel Divide-and-Conquer

I-Chen Wu and H. T. Kung *

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

This paper studies the relationship between parallel computation cost and communication cost for performing divide-and-conquer (D&C) computations on a parallel system of p processors. The parallel computation cost is the maximal number of the D&C nodes that any processor in the parallel system may expand, whereas the communication cost is the total number of cross nodes. A cross node is a node which is generated by one processor but expanded by another processor. A new scheduling algorithm is proposed, whose parallel computation cost and communication cost are at most $\lceil N/p \rceil$ and pdh, respectively, for any D&C computation tree with N nodes, height h, and degree d. Also, lower bounds on the communication cost are derived. In particular, it is shown that for each scheduling algorithm and for each positive $\epsilon_C < 1$, which can be arbitrarily close to 0, there are values of N, h, d, p, and $\epsilon_T (> 0)$, for which if the parallel computation cost is between N/p (the minimum) and $(1+\epsilon_T)N/p$, then the communication cost must be at least $(1-\epsilon_C) \cdot pdh$. Therefore, the proposed scheduling algorithm is optimal with respect to the communication cost, since the parallel computation cost of the algorithm is near optimal.

1 Introduction

Divide and conquer (D&C) is a common computation paradigm, in which the solution to a problem is obtained by solving subproblems recursively. Examples of D&C computations include various sorting methods such as quick sort [6], computational geometry procedures such as convex hull calculation [12], AI search heuristics such as constraint satisfaction techniques [5], adaptive data classification procedures such as generation and maintenance of quadtrees [13], and numerical methods such as multigrid algorithms [10] for solving partial differential equations.

A D&C computation can be viewed as a process of expanding and shrinking a tree. Each node in the tree corresponds to a problem instance, and children of the node correspond to its subproblems. During the computation, each internal (non-leaf) node goes through two phases. The first phase is the *divide phase* during which the problem instance associated with the node is divided into subproblems. The second phase is the *combine phase* during which the solution of the problem instance associated with the node is derived by combining solutions of the subproblems associated with the node's children. After its creation each leaf will perform some computation and return the results to its parent. At a given time, nodes on a wavefront that cuts across all paths from the root to leaves can be active in performing divide, combine, or compute operations. Along each path the wavefront first moves down from the root to its leaf and then up from the leaf to the root.

At first glance, one might think that it should be straightforward to perform D&C in parallel, because nodes on the wavefront can all be processed independently. However, if one wants to achieve good load balancing between the processors, then parallelizing D&C becomes nontrivial. In fact, doing efficient D&C on any real parallel machine has been a major challenge to researchers [3, 4, 9, 14] for many years.

The difficulties are due to the fact that many D&C computations are highly dynamic in the sense that these computations are data-dependent. During computation, a problem instance can be expanded into any number of subproblems depending on the data that have been computed so far. In fact, the trees of

*This research was supported in part by the Defense Advanced Research Projects Agency (DOD) monitored by DARPA/CMO under Contract MDA972-90-C-0035, in part by the National Science Foundation and the Defense Advanced Research Projects Agency under Cooperative Agreement NCR-8919038 with the Corporation for National Research Initiatives, and in part by the Office of Naval Research under Contract N00014-90-J-1939.

CH3062-7/91/0000/0151$01.00 © 1991 IEEE

many D&C computations can be expected to be sparse and irregular, and as a result, load balancing must be adaptive to the tree structure and must be done dynamically at run time. This implies that computation loads need to be moved around between processors during computation. The challenge is then to devise efficient scheduling algorithms which can achieve good load balancing while minimizing the communication cost for moving computations around.

In general there is a tradeoff between balancing computation loads and minimizing communication costs. The results of this paper quantify this tradeoff. In particular, the paper establishes lower bounds on the communication cost for any scheduling algorithm based on how well it performs load balancing.

2 Summary of Results of This Paper

2.1 Definitions and Notation

The tree of a D&C computation is called a (N, h, d)-*tree*, if

- N is the number of nodes in the tree,
- h is the height of the tree, and
- d is the maximal number of children of a node. (We assume that d is at least 2, to allow parallel processing of the tree.)

A node is said to be at tree level i if it is the i-th node on the path from the root to the node. Therefore, the root is at level 1, and the height of the tree is the maximal level number.

For the parallel system which will carry out the D&C computation, we assume that

- p is the number of processors in the system, and
- it takes one time step for a processor to *expand* a node, i.e., to perform the divide operation for an internal node, or to perform the compute operation for a leaf node. For simplicity, we assume that a processor takes no time to perform a combine operation.

When a node is expanded, zero or more children may be *generated*. More precisely, if a node does not generate any children, the node is a leaf; if a node generates one or more (up to d) children, the node is an internal node. Each newly generated node will in turn be expanded by some processor in the future. A *frontier node* is a node which has been generated but has not been expanded.

A *scheduling algorithm* for a D&C computation schedules nodes (i.e., frontier nodes) on processors for expansion. We assume that scheduling algorithms cannot "lookahead". This non-lookahead assumption is reasonable when dealing with irregular D&C trees. In this type of tree, the number of children a parent may have (if any) is typically data-dependent and is therefore not known *a priori*.

The *parallel computation cost* $T_{\mathcal{A}}(H)$ of a scheduling algorithm $\mathcal{A}$ for a D&C computation tree H is the maximum number of the nodes that any processor may expand. Since there are N nodes and p processors, a lower bound on $T_{\mathcal{A}}(H)$ is $T_{min} = \lceil N/p \rceil$. The parallel computation cost $T_{\mathcal{A}}$ of algorithm $\mathcal{A}$ is defined as the maximum $T_{\mathcal{A}}(H)$ for all (N, h, d)-trees H.

The *communication cost* $C_{\mathcal{A}}(H)$ of a scheduling algorithm $\mathcal{A}$ for a D&C computation tree H is the total number of cross nodes. A *cross node* is a node which is generated by one processor but expanded by another processor. Note that the processor expanding a cross node needs to receive information from the processor generating the node. Therefore, $C_{\mathcal{A}}(H)$ is a reasonable measure for capturing the interprocessor communication cost in performing the divide phase of all the internal nodes. (A similar definition of communication cost is used by Papadimitriou and Ullman in [11].) The communication cost $C_{\mathcal{A}}$ of algorithm $\mathcal{A}$ is defined as the maximum $C_{\mathcal{A}}(H)$ for all (N, h, d)-trees H.

2.2 Main Results

Theorem 1 *For each scheduling algorithm $\mathcal{A}$ for a parallel system of p processors, for each integer p', $0 < p' \leq p$, and for each N, h, and d with the following two restrictions,*

S1. $N > 3pd^2h$, *and*

S2. $h > \lceil \log_d N \rceil + \lceil \log_d pdh \rceil + 1$,

there exists some (N, h, d)-tree H for which at least one of the following two properties is true:

P1. *the parallel computation cost of the algorithm is* $T_{\mathcal{A}}(H) \geq N'/p'$;

P2. *the communication cost of the algorithm is* $C_{\mathcal{A}}(H) \geq C'$,

where $N' = N - 3pd^2h$, $C' = p'\kappa$, $\kappa = (d-1)h'$, *and* $h' = h - \lceil \log_d N \rceil - \lceil \log_d pdh \rceil - 1$.

Many D&C computations are expected to satisfy restrictions S1 and S2. Since N is usually an exponential function of h, restriction S1 is easily satisfied in these cases. Restriction S2 roughly requires that $N < d^{h-2}/ph$. If a tree is perfectly balanced and each node has exactly d children, then N would be $\Theta(d^{h-1})$ instead. A perfectly balanced tree is easy for load balancing because the subtrees of each node have the same computation load. Restrictions S1 and S2 basically capture those interesting D&C computations with irregular trees. This class of D&C computations are exactly those for which one finds it difficult to achieve good load balancing without paying much in communication overheads. The lower bound on $C_{\mathcal{A}}(H)$, stated in P2 of the theorem, provides an explanation of why this must be the case.

The two properties P1 and P2 in Theorem 1 can be expressed in terms of the quantities N, h, d (associated with the D&C tree) and p (associated with the parallel system) as follows. One can check that $N' \geq (1-\epsilon_N)N$ and $h' \geq (1-\epsilon_h)h$ for each positive $\epsilon_N \leq 1$ and $\epsilon_h \leq 1$, provided that $h \geq \frac{2\log_d ph+\log_d 3+6-\log_d \epsilon_N}{\epsilon_h}$ and $\frac{d^{\epsilon_h h}}{pd^4h} \geq N \geq \frac{3pd^2h}{\epsilon_N}$. (Note: if $h \geq \frac{2\log_d ph+\log_d 3+6-\log_d \epsilon_N}{\epsilon_h}$, then $\frac{d^{\epsilon_h h}}{pd^4h} \geq \frac{3pd^2h}{\epsilon_N}$; if $\frac{d^{\epsilon_h h}}{pd^4h} \geq N$, then $\epsilon_h h \geq \log_d N + \log_d pdh + 3 \geq \lceil \log_d N \rceil + \lceil \log_d pdh \rceil + 1 = h - h'$, i.e., $h' \geq (1-\epsilon_h)h$; if $N \geq \frac{3pd^2h}{\epsilon_N}$, then $N' \geq (1-\epsilon_N)N$.) From this and the fact that $N' < N$ and $h' < h$, we note that N' and h' approach N and h respectively, when both ϵ_N and ϵ_h approach 0. Therefore, P1 and P2 in Theorem 1 become $T_{\mathcal{A}}(H) = \Omega(N/p)$ and $C_{\mathcal{A}}(H) = \Omega(pdh)$ for large h, when p' is close to p. Furthermore, we can slightly change the theorem as Corollary 1.

Corollary 1 *For each scheduling algorithm for a parallel system of p processors, for each positive $\epsilon_C < 1$, which can be arbitrarily close to 0, there are values of N, h, d, p, and $\epsilon_T(> 0)$, for which if the parallel computation cost is between $\frac{N}{p}$ and $(1+\epsilon_T)\frac{N}{p}$, then the communication cost must be at least $(1-\epsilon_C)C_u$, where $C_u = pdh$.*

Proof. Let $p \geq \frac{3}{\epsilon_C}$ and $d \geq \frac{3}{\epsilon_C}$. Then, let $\epsilon_T = \frac{1}{2p}$. And, let N and h be in the range as shown above with $\epsilon_h = \frac{\epsilon_C}{3}$ and $\epsilon_N = \frac{1}{2p}$. One can check that $(1+\epsilon_T)\frac{N}{p} \leq \frac{(1+\epsilon_T)N'}{p(1-\epsilon_N)} = \frac{N'(2p+1)}{p(2p-1)} < \frac{N'}{p-1}$ and $p'(d-1)h' \geq (1-\frac{1}{p})(1-\frac{1}{d})(1-\epsilon_h)C_u \geq (1-\frac{\epsilon_C}{3})^3 C_u > (1-\epsilon_C)C_u$ when $p' = p-1$. Thus, if $\frac{N}{p} \leq T_{\mathcal{A}} \leq (1+\epsilon_T)\frac{N}{p}$ $(< \frac{N'}{p-1})$, the communication cost must be at least $(1-\epsilon_C)C_u$. □

Theorem 1 also implies an important tradeoff result: if a scheduling algorithm wants to achieve a good load balancing by parallel processing, then it must pay a high price in communication cost. We can express the tradeoff between $T_{\mathcal{A}}$ and $C_{\mathcal{A}}$ explicitly by showing a lower bound on their product: $T_{\mathcal{A}} \cdot (C_{\mathcal{A}} + \kappa)$. If $(p^* - 1)\kappa \leq C_{\mathcal{A}} < p^*\kappa$, where $0 < p^* \leq p$, then by Theorem 1, $T_{\mathcal{A}}$ must be at least N'/p^*. Therefore, $T_{\mathcal{A}} \cdot (C_{\mathcal{A}} + \kappa) \geq (N'/p^*) \cdot p^*\kappa = N' \cdot \kappa$. Note that because of $T_{\mathcal{A}} \geq N/p \geq N'/p$ this tradeoff is also satisfied when $C_{\mathcal{A}} \geq p\kappa$. This tradeoff result is summarized in the following corollary.

Corollary 2 *For any scheduling algorithm $\mathcal{A}$ for a parallel system of p processors, for all N, h, and d with restrictions S1 and S2 as defined in Theorem 1,*

$$T_{\mathcal{A}} \cdot (C_{\mathcal{A}} + \kappa) \geq N' \cdot \kappa,$$

where N' and κ are defined in Theorem 1.

Theorem 2 *A scheduling algorithm $\mathcal{A}$ can be devised to have the property that the parallel computation cost is $T_{\mathcal{A}} = T_{min}$ and the communication cost is $C_{\mathcal{A}} \leq C_u(= pdh)$ for any (N, h, d)-tree.*

The algorithm satisfying Theorem 2 has the minimum parallel computation cost. By Corollary 1, the algorithm is optimal with respect to the communication cost, since the parallel computation cost of the algorithm is near optimal. These results also imply that the lower bound on $T_{\mathcal{A}} \cdot (C_{\mathcal{A}} + \kappa)$ in Corollary 2 is tight when both ϵ_N and ϵ_h are arbitrarily close to 0.

Note that Theorems 1 and 2 are so formulated that their results are *system-independent.* That is, the results are independent from the interconnection topology of the processors and various control overheads such as data structure maintenance and reading/writing messages. Therefore, our upper and lower bounds on $C_{\mathcal{A}}$ are intrinsic to any parallel system. These bounds give insights into actual communication cost in a real implementation, but exactly how they are related to the actual cost is a separate matter depending on the implementation (see [15]).

Section 3 describes the algorithm of Theorem 2. Section 4 presents a simplified version of Theorem 1 and its proof to help the reading of this paper. A complete proof of Theorem 1 is given in Section 5.

2.3 Relation to Past Work

There have been several approaches in performing parallel D&C. A simple approach (e.g., in [2]) is to

expand all the nodes above a fixed level on one processor and then distribute nodes at this level to other processors. Load balancing would be done poorly in this approach when the tree is irregular. Another approach [14] is to distribute generated nodes, and to have each processor perform load balancing based on load status information from its neighbor processors. For this scheme, the communication cost can be very high in the worst case.

Recently, some researchers have made efforts to reduce communication overhead. A popular approach [4, 9, 16] is based on the "donate-highest-subtree" strategy, in which an idle processor will be given frontier nodes as near to the root as possible. Since a subtree rooted near the top usually has many nodes and these nodes can all be expanded locally, this strategy tends to reduce the amount of interprocessor communication. Ferguson and Korf [3] presented a D&C scheme with several processors scheduled first to a node and then to their children. The idea behind their scheme is also that of distributing frontier nodes near the root to idle processors.

Although the methods described in the previous paragraph all attempt to reduce communication overhead, they do not use global information to balance the load. It turns out that the communication cost for these methods can still be high in the worst case. For example, we estimate that the communication cost is $O(dh^{\log_d p})$ for Ferguson and Korf's scheme, and is $O(\min(p^2h, pdh^2))$ for the scheme in [4] with round-robin scheduling.

In contrast, the communication cost for the scheduling algorithm of this paper (Section 3) is as low as $O(pdh)$ (Theorem 2). This is partly due to the fact that our algorithm is able to make effective use of global information (i.e., "global pool" in Section 3).

Most importantly, we note that none of the previous work has any lower bound results on the communication cost for parallel D&C computations. It appears that our lower bounds in Theorem 1 and Corollaries 1 and 2 are the first lower bound results for those D&C computations whose tree structures are dynamic in the sense that the tree structure is determined only at run time. Previous results on computation and communication cost tradeoffs such as those in [7, 8, 11] deal with only *static* computation graphs, whose topologies are known before the computation starts.

3 A Scheduling Algorithm and Upper Bounds

This section describes a new scheduling algorithm which can achieve the upper bounds in Theorem 2 for both parallel computation cost and communication cost. The bounds hold for any D&C computation, i.e., for any (N, h, d)-tree no matter how irregular it is.

Proposed Scheduling Algorithm

The scheduling algorithm uses a data structure, called a *Global Pool* (abbr. *GP*), to keep track of frontier nodes at a particular tree level which have not been taken by any processor for expansion. This level, identified by a variable gl, has the property that nodes at higher levels have all been taken by processors. Every processor will try to take a node from the GP to work on whenever it becomes idle. For the proof of Theorem 2, it suffices to assume that the GP is maintained by some single processor. (See [15] for a distributed scheme where the GP is maintained by multiple processors.)

Initially, the GP contains only the root and the value of gl is one. The GP becomes empty when all of its nodes at level gl have been taken by the processors. At this moment, all the processors are requested to send in their frontier nodes at level $gl+1$ in the next time step when all the nodes at level $gl+1$ have been generated. Then the GP is filled with this set of new nodes, and gl is increased by one. This process is repeated until all the nodes have been expanded.

The key idea of this algorithm is what each processor will do after it has taken a node from the GP. The processor will do a depth-first traversal. Consequently, the processor can exhaust all possible work locally before asking for a new node from the GP. As a result, we can prove (below) that the communication cost can be as low as C_u. While not related to parallel computation cost and communication cost, an important advantage of this local depth-first strategy is that it uses the minimum amount of memory.

In essence the scheduling algorithm described here uses a breadth-first scheme to distribute big chunks of computations to processors, and has each processor after receiving a computation follow the depth-first strategy locally. Therefore, the algorithm is a hybrid method, which interestingly will do a purely depth-first traversal of the tree in the case that only one processor is used.

Suppose that we define the *parallel computation time* to be the time (in terms of number of time steps) when the last node is expanded by a processor. Then

the parallel computation time of the algorithm described here is at most $\lceil N/p + h \rceil$. To see this, we note that some processors may become idle only when the number of nodes in the GP is smaller than the number of idle processors. In the worst case all the p processors may become idle at the end of some time step, but at this time there is only one node in the GP. Thus, in the next time step, as many as $p-1$ processors may be idle. This situation can happen at most h times. Therefore, in the entire D&C computation, additional $h(p-1)$ nodes could have been expanded if there were no idle processors at any time step. This implies that the parallel computation time is at most $\lceil (N + h(p-1))/p \rceil \leq \lceil N/p + h \rceil$.

Note that parallel computation time defined in the previous paragraph is different from parallel computation cost defined in Section 2.1. Being able to take into account processor waiting time induced by internode dependency, parallel computation time may be of more practical interest than parallel computation cost.

However, to prove Theorem 2, we need to establish an upper bound on the parallel computation cost of the algorithm. We will do this and also establish an upper bound on the communication cost of the algorithm.

Proof of Theorem 2. To achieve the $\lceil N/p \rceil$ upper bound on parallel computation cost, we will need to add some fair scheduling feature to the algorithm described above. Whenever the number of nodes in the GP is smaller than the number of idle processors, we will select the active processors for the next time step from *all* the p processors in a fair way. That is, processors take turn to become active using a round-robin scheme. This ensures at the end of any time step that the total number of nodes expanded by a processor so far will not exceed that expanded by any other processor by more than one. Thus when all the N nodes are expanded, each processor will have expanded at most $\lceil N/p \rceil$. This proves that the parallel computation cost of the scheduling algorithm with the fair scheduling feature is at most $\lceil N/p \rceil$.

The communication cost of the algorithm is at most the number of frontier nodes entering the GP, as this represents the only interprocessor communication activity for the entire algorithm. Since by using depth-first search each processor has at most d local nodes at each level (as illustrated in Figure 1), the GP can collect at most pd nodes each time that gl increases. This will happen at most h times, so the total number of nodes entering the GP is bounded above by $C_u = pdh$. □

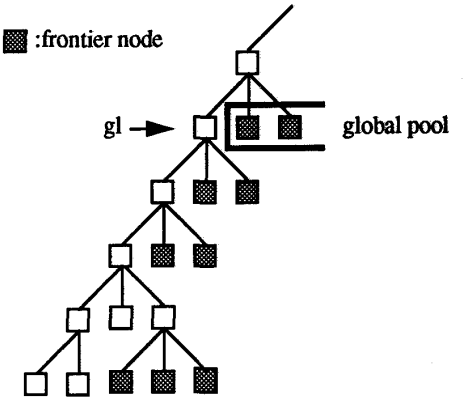

Figure 1: At most d frontier nodes at each level on a processor ($d = 3$).

Note that in a practical implementation, the fair scheduling feature may not be used since minimizing parallel computation cost may not be important. Without the fair scheduling feature, the parallel computation cost would become $\lceil N/p + h \rceil$. However, the communication cost can be reduced to $p(d-1)h$, if a processor right after expanding a node will schedule one child, if any, of the node for expansion at the next time step.

The scheduling algorithm described in this section is being used as a basis for developing a parallel programming model for D&C computations. To obtain practical insights, we plan to implement a programming system based on the model on the 26-host Nectar network system [1] developed at Carnegie Mellon University.

4 A Simplified Version of Theorem 1

This section presents Theorem 3 (see below), which is a simplified version of Theorem 1 dealing with only two processors. A relatively simple proof of Theorem 3 is given. This simple proof captures the essence of a more complicated proof of Theorem 1 given in Section 5. It is advised that the reader read this simple proof first to understand the ideas.

Theorem 3 *For each scheduling algorithm $\mathcal{A}$ for a parallel system of two processors, for each N, h, and d with the following three restrictions,*

S1. $N > 3dh$,

S2. $h > \lceil \log_d N \rceil + 2$, *and*

S3. $h - \lceil \log_d N \rceil - 2$ *is an even integer,*

there exists some (N, h, d)*-tree* H *for which at least one of the following two properties is true:*

Q1. *the parallel computation cost of the algorithm is* $T_{\mathcal{A}}(H) \geq N - 3dh$*;*

Q2. *the communication cost of the algorithm is* $C_{\mathcal{A}}(H) \geq h'(d-1)$*,*

where $h' = (h - \lceil \log_d N \rceil - 2)/2$.

Note that restrictions S1 and S2 correspond to those in Theorem 1. Restriction S3 is for a minor technical convenience, namely, ensuring that h' an integer.

Theorem 3 implies, for example, that if the communication cost is small (in the sense that Q2 does not hold), then the parallel computation cost must be large (in the sense that Q1 holds). In particular, if $C_{\mathcal{A}}(H) < h'(d-1)$ and if $3dh \ll N$, then the parallel computation cost will be close to N.

Proof of Theorem 3. Suppose that we are given a scheduling algorithm $\mathcal{A}$ for performing a D&C computation on processors P_1 and P_2. For algorithm $\mathcal{A}$, we will prove the existence of a (N, h, d)-tree H for which at least one of Q1 and Q2 must hold.

By playing an adversary game with algorithm $\mathcal{A}$, we will construct the tree by growing it from the root one step at a time. A time step consists of two phases, node scheduling phase and node expansion phase. In the node scheduling phase, algorithm $\mathcal{A}$ schedules a node or no node for each processor to execute. Then, in the node expansion phase, these scheduled nodes are expanded. In this phase we will determine the number of children each scheduled node will generate.

We will first define a special class of subtrees which will be used to describe some sufficient conditions under which a tree can grow to a (N, h, d)-tree. We will then give the main part of the proof including a description of the tree construction procedure.

HFD-Subtree

Definition 1 *At any given time during the tree construction, a High-and-Full-Degree subtree (abbv. HFD-subtree) is a subtree, which is rooted at a node at or above level* $h - \lceil \log_d N \rceil$*, and which has been constructed using the following rules:*

A1. *nodes above level* h *generate* d *children; and*

A2. *nodes at level* h *generate no children.*

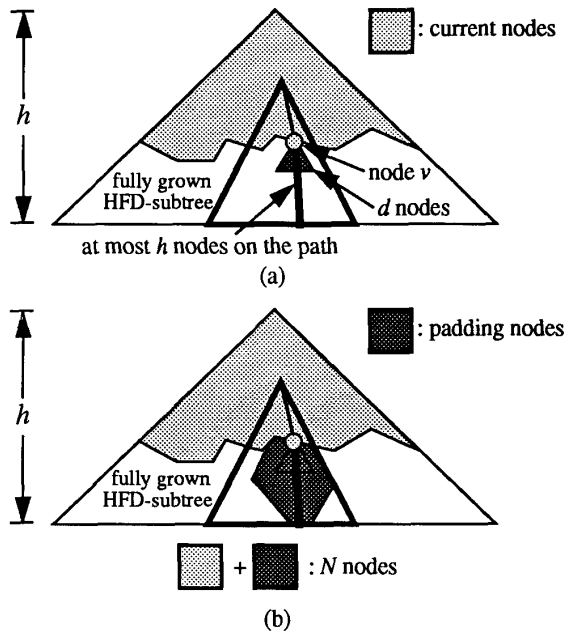

Figure 2: Growing the current tree to a (N, h, d)-tree.

Note that rules A1 and A2 imply that a node which is above level h and has no children must be a frontier node.

Lemma 1 *At any given time during the tree construction, if the current tree satisfies the following four properties:*

I1. *the total number of generated nodes is at most* $N - h - d$ *(generated nodes include the root);*

I2. *the height is at most* h*;*

I3. *the degree of any node is at most* d*; and*

I4. *the tree contains an HFD-subtree,*

then a construction procedure can be devised to grow the tree to a (N, h, d)*-tree:*

Proof. We first note that in the HFD-subtree of I4 there exist nodes which are above level h and have no children. Otherwise, the subtree would have been "fully grown" to level h, according to rules A1 and A2. Since its root is at and above level $h - \lceil \log_d N \rceil$, this fully grown HFD-subtree would have at least $d^{\lceil \log_d N \rceil} (\geq N)$ nodes. This contradicts I1. As noted above, those nodes in the current HFD-subtree which are above level h and have no children must all be frontier nodes.

Let H_1 be the current tree. We will identify a set of "padding nodes" which can be added to H_1 to make it a (N, h, d)-tree.

If H_1 has height less than h or degree less than d, we will grow it by extending the current HFD-subtree from one of its frontier nodes which are above level h. Let v be this frontier node, as shown in Figure 2. We generate d children for v and create a path from v to a node at level h, as shown in Figure 2 (a). The resulting tree, called H_2, has height h, degree d, and no more than $(N-h-d)+d+h = N$ nodes.

If H_2 has less than N nodes, we will pad it with nodes in the fully grown HFD-subtree which are reachable from the current frontier nodes and other padding nodes, as illustrated in Figure 2 (b). Since the fully grown HFD-subtree has at least N nodes, it has sufficient nodes which can be added to H_2 to make it a (N, h, d)-tree.

After having identified all these padding nodes, we now have a "blueprint" for a construction procedure to follow. More precisely, the construction procedure will just generate all those padding nodes in the dark region in Figure 2 (b). □

Main Part of Proof of Theorem 3

The tree construction procedure consists of three stages. Each stage uses an independent set of rules in constructing the tree.

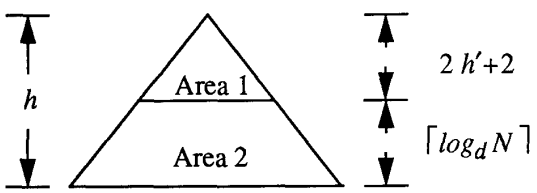

Figure 3: Two areas in the constructed tree.

In stage 1, we expand each node with exactly d children. Stage 1 terminates at time T_1 when a total of $2h'$ or $2h'+1$ nodes have just been expanded. (Note that at this time the tree is completely inside area 1 of Figure 3.) Since the number of frontier nodes increases by $d-1$ each time when a node is expanded, there are exactly $2h'(d-1)+1$ or $(2h'+1)(d-1)+1$ frontier nodes at time T_1. Without loss of generality, we assume that processor P_1 has generated at least $h'(d-1)$ frontier nodes.

Stage 2 starts right after T_1. In this stage every node above level h expanded by processor P_1 will have d children, while every node at level h or expanded by processor P_2 will have no children. Stage 2 terminates at time T_2 when one of the following two conditions becomes true:

C1 At least $h'(d-1)$ cross nodes have been scheduled.

C2 At least $N-h-2d$ nodes have been generated.

The following shows that C1 or C2 must become true sometime, i.e., T_2 exists. Recall that by the end of stage 1 processor P_1 has generated at least $h'(d-1)$ frontier nodes. In stage 2 processor P_1 will generate nodes in the subtrees rooted at those frontier nodes which are still in P_1. For each of these subtrees, since its root is in area 1 of Figure 3, the subtree can have at least $N-h-2d$ nodes unless some of these nodes are moved to processor P_2 from processor P_1. If C1 does not hold, then fewer than $h'(d-1)$ nodes can be moved from P_1 to P_2. Consequently, some subtree will have at least $N-h-2d$ nodes, and thus C2 will be true.

Stage 3 starts right after time T_2. Lemma 2 below shows that properties I1-I4 of Lemma 1 hold for the tree at time T_2. In stage 3, we follow the procedure described in the proof of Lemma 1 to grow the tree to a (N, h, d)-tree.

Lemma 2 *At any time in stage 1 or 2, including time T_2, the tree satisfies properties I1-I4 of Lemma 1.*

Proof. It is obvious from the descriptions of stages 1 and 2 that I2 and I3 are satisfied. For I1, we note that the total number of nodes generated in stage 1 is at most $(2h'+1)d+1$, and thus at most $N-h-d$ by restriction S1 of Theorem 3. In stage 2, I1 obviously holds when C2 is not true. Suppose that C2 becomes true at time T_2. Since the tree has no more than $N-h-2d$ nodes in the previous time step and since at most d nodes can be generated (in processor P_1) in one time step, there are at most $N-h-d$ nodes at time T_2.

Property I4 clearly holds for stage 1 by examining its description. It remains to prove that I4 holds for stage 2. The proof is similar to the earlier proof of the fact that C1 or C2 must become true in stage 2. Recall that in stage 1 processor P_1 has generated at least $h'(d-1)$ frontier nodes. We note that any of these subtrees rooted at these nodes is an HFD-subtree if the subtree does not contain any expanded cross node. Since the number of cross nodes expanded (not just scheduled) through time T_2 is less than $h'(d-1)$, one of these subtrees must be an HFD-subtree. Note that if C2 becomes true at time T_2 (in the node scheduling

phase), the node scheduled has not been expanded. □

To complete the proof of Theorem 3, we observe that if C1 becomes true at some time in stage 2 or 3, it will remain true for the rest of the tree construction process. Therefore property Q2 of Theorem 3 will hold for the final (N, h, d)-tree.

Now assuming that C1 never holds at any time in stage 2 or 3, we want to show that property Q1 of Theorem 3 will hold for the final (N, h, d)-tree. We derive an upper bound on the total number of nodes expanded by processor P_2. The upper bound is the sum of four terms U_1, U_2, U_3 and U_4. In stage 1, processor P_2 has expanded at most $U_1 = 2h'+1$ nodes. At time T_1, processor P_2 can have generated up to $(h'+1)(d-1)+1$ frontier nodes, each of which can be expanded at most once by processor P_2 in stage 2 or 3. It is also possible for processor P_2 to expand nodes which are generated by P_1 but subsequently moved to P_2. The total number of these nodes is at most $C_{\mathcal{A}}(H) \leq U_3 = h'(d-1)$. Moreover, to take care of the nodes generated after T_2 in stage 3, processor P_2 may expand up to $U_4 \leq h + 2d$ nodes. Therefore the total number of nodes expanded by processor P_2 is at most $U = U_1 + U_2 + U_3 + U_4 \leq 3dh$. This implies that processor P_1 has expanded at least $N - U = N - 3dh$; that is, property Q1 holds. □

5 Proof of Theorem 1

Suppose that we are given a scheduling algorithm $\mathcal{A}$ for performing a D&C computation on a parallel system of p processors. For algorithm $\mathcal{A}$, we will prove the existence of a (N, h, d)-tree H for which either only p' processors are active for expanding most of nodes (at least N' nodes) or at least C' nodes are moved between processors to balance their computation loads. For the former, the parallel computation cost will be high, i.e., $T_{\mathcal{A}}(H) \geq N'/p'$ (property P1). For the latter, the number of cross nodes will be large, i.e., $C_{\mathcal{A}}(H) \geq C'$ (property P2).

By playing an adversary game with algorithm $\mathcal{A}$, we will construct the tree by growing it from the root one step at a time. The definition of time step is the same as that in the proof of Theorem 3.

We will give some more definitions in Section 5.1 and then give the main part of this proof in Section 5.2. All the related lemmas are in Section 5.3.

5.1 Definitions

To help derive a lower bound on the number of cross nodes, we introduce the following relation between subtrees.

Definition 2 *A set of subtrees is processor-or-ancestry independent (abbr. PA-independent) if for each pair of subtrees in the set at least one of the following two properties is satisfied:*

1. *Processor Independence: the roots of these two subtrees are generated on different processors;*
2. *Ancestry Independence: neither is a subtree of the other. That is, there is no ancestor-descendant relationship between the two roots.*

Note that for two PA-independent subtrees rooted at nodes r_1 and r_2, if node r_1 is an ancestor of node r_2, then both nodes must be generated on different processors. This implies that there must exist at least one cross node on the path from node r_1 (inclusive) to the parent (inclusive) of node r_2. Therefore, from this property, if there are k PA-independent subtrees each of which has at least one expanded cross node, then there are at least k expanded cross nodes in the tree. This is shown in Lemma 3 (in Section 5.3).

Definition 3 *An HFDC-subtree is an HFD-subtree (as defined in Definition 1) or a subtree with at least one cross node already expanded. If the root of an HFDC-subtree is generated on processor P, the subtree is called an HFDC-subtree on processor P.*

By Lemma 3 and Definition 3, if there are k PA-independent HFDC-subtrees and fewer than k expanded cross nodes, then there exists an HFD-subtree, as shown in Lemma 4. We will use this lemma to show the existence of an HFD-subtree during some periods of the tree construction procedure.

5.2 Main Part of Proof of Theorem 1

The tree construction procedure, like that in Section 4, consists of three stages. Basically, this procedure, summarized in Figure 4, is similar to that in Section 4, except that in stage 1 we use more sophisticated rules to prove a better lower bound of the number of cross nodes. (Note that if $h \gg \log_d N$ and $p = 2$ the lower bound of communication cost in this theorem is approximately twice as large as that in Theorem 3.)

In stage 1, we will repeatedly apply rules R1-R4 (in Figure 4) until time T_1 when one of the conditions

Stage 1 $\Longrightarrow$

Apply the following four rules:

R1. Nodes in area 1 (shown in Figure 5) will generate d children.

R2. Cross nodes in areas 2 and 3 (shown in Figure 5) will not generate any children.

R3. Non-cross nodes in areas 2 and 3 (excluding level h) will generate d children.

R4. Nodes at level h will not generate any children.

Repeat rules R1-R4 **until** time T_1 when any of the following three conditions holds:

C1. For some p' processors, at least h' non-cross nodes have been expanded on each processor.

C2. At least C' cross nodes have been scheduled.

C3. At least $N - (pd + d + h)$ nodes have been generated.

Stage 2 (continued from time T_1 when C1 holds) $\Longrightarrow$

Find a set Γ of p' processors with the following two properties:

B1. There are at least C' PA-independent HFDC-subtrees in Γ.

B2. There are at most h' non-cross nodes expanded on each of the other $p - p'$ processors in the set $\overline{\Gamma}$.

Apply the following three rules:

R5. Nodes (excluding those at level h) in Γ will generate d children.

R6. Nodes in $\overline{\Gamma}$ will not generate any children.

R7. Nodes at level h will not generate any children.

Repeat rules R5-R7 **until** time T_2 when either of the following two conditions holds:

C4. At least C' cross nodes have been scheduled.

C5. At least $N - (pd + d + h)$ nodes have been generated.

Stage 3 (continued from time T_1 when C2 or C3 holds or from time T_2 when C4 or C5 holds.) $\Longrightarrow$

Use the construction procedure described in the proof of Lemma 1 to grow the tree to a (N, h, d)-tree.

Figure 4: Tree construction procedure.

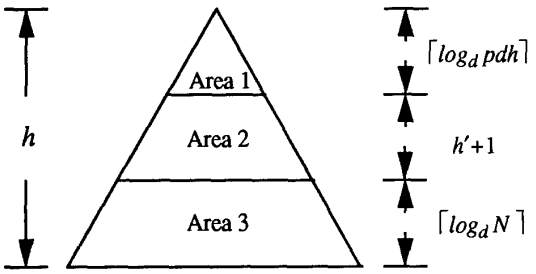

Figure 5: Three areas in the constructed tree.

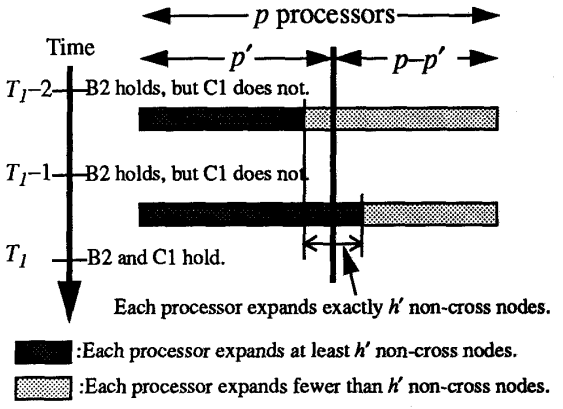

Figure 6: Around the time when condition C1 becomes true.

C1-C3 holds. Rules R1-R4 ensure that each subtree rooted in area 1 or 2 is always an HFDC-subtree because in constructing the subtree either rules A1 and A2 are followed (using R1, R3, and R4) or some cross nodes are expanded (using R2). Basically, the procedure in stage 1 attempts to produce at least C' PA-independent HFDC-subtrees on some p' processors (property B1) while preventing each of the other $p - p'$ processors from expanding more than h' non-cross nodes (property B2). (Recall that in the proof of Theorem 3 subtrees rooted at frontier nodes at time T_1 are PA-independent HFDC-subtrees.)

If condition C1 holds at time T_1, then from Figure 6 we can find a set Γ of p' processors for which condition C1 and property B2 hold. According to Lemma 5, there are at least $\kappa(= (d-1)h')$ PA-independent HFDC-subtrees on each processor which has expanded h' non-cross nodes. So there are at least $C'(= p'\kappa)$ PA-independent HFDC-subtrees in Γ at this time. Therefore, property B1 holds, and we are ready for stage 2.

In stage 2, we will repeatedly apply rules R5-R7 until time T_2 when condition C4 or C5 holds. (Note that these rules are exactly the same as those of stage 2 in Section 4.) According to property B1, initially,

there are at least C' PA-independent HFDC-subtrees in Γ. In stage 2, these subtrees continue to be HFDC-subtrees, because either rules A1 and A2 are followed (using R5 and R7) or some cross nodes are expanded (using R6). In addition, by rule R6, the set $\overline{\Gamma}$ of the other $p - p'$ processors will not generate any new nodes.

Now, we want to show that one of the conditions C2-C5 must become true at time T_1 or T_2. According to Lemma 6 (in Section 5.3), at any time in stage 1 or 2 properties I1-I4 of Lemma 1 hold; so, at any time in stage 1 or 2 the tree will be able to grow to a (N, h, d)-tree by Lemma 1. Hence, if C2 or C4 never hold, C3 or C5 becomes true.

Stage 3 starts right after one of the conditions C2-C5 becomes true. (If C2 or C3 holds at T_1, this implies that stage 2 is empty.) Since Lemma 6 also shows that properties I1-I4 of Lemma 1 hold for the tree at time T_1 or T_2, in stage 3 we will follow the procedure described in the proof of Lemma 1 to grow the tree to a (N, h, d)-tree H.

To complete the proof, we observe that if $C_{\mathcal{A}}(H) \geq C'$ it will remain true for the rest of the tree construction process. Therefore property P2 of Theorem 3 will hold for H.

Now, assuming that $C_{\mathcal{A}}(H) < C'$, we want to prove that property P1 holds for H. Since C2 and C4 never hold, either C3 will become true at time T_1 or C5 will become true at time T_2. First, suppose that condition C5 becomes true at time T_2. To prove that property P1 holds in this case, we will derive an upper bound on the total number of nodes expanded in $\overline{\Gamma}$. The upper bound consists of five terms U_1, U_2, U_3, U_4, and U_5. Assume that there are $C_1 < U_1 = C'$ cross nodes expanded in $\overline{\Gamma}$ in stage 1. In stage 1, the processors in $\overline{\Gamma}$ have expanded at most $U_2 = (p - p')h'$ non-cross nodes due to property B2. These nodes expanded in stage 1 will generate at most $U_3 = ((p - p')h' + C_1)d$ frontier nodes in $\overline{\Gamma}$ at time T_1, each of which can be expanded at most once in $\overline{\Gamma}$. After time T_1, it is also possible for the processors in $\overline{\Gamma}$ to expand nodes moved from the processors in Γ. The total number of these nodes is $U_4 \leq C_{\mathcal{A}}(H) - C_1$. Moreover, to take care of the nodes generated after T_2, processors in $\overline{\Gamma}$ may expand up to $U_5 \leq pd + d + h$ nodes. Therefore, the total number of nodes expanded in $\overline{\Gamma}$ is at most $U = U_1 + U_2 + U_3 + U_4 + U_5 \leq 3pd^2h$. This implies that the processors in Γ have expanded at least $N - U = N - 3pd^2h$ nodes; therefore, $T_{\mathcal{A}}(H) \geq (N - 3pd^2h)/p' \geq N'/p'$, i.e., property P1 holds.

Suppose that condition C3 becomes true at time T_1. Since condition C1 does not hold in stage 1, we can find a set Γ of p' processors with property B2 (see Figure 6 also). Since stage 2 is empty for this case, we can let time T_2 be the same as T_1. Thus, we can use the same technique as above to prove that property P1 holds. □

5.3 Relevant Lemmas

Lemma 3 *Suppose that there are k PA-independent subtrees at some time during the computation. If each of these subtrees has at least one expanded cross node, then the total number of expanded cross nodes in the whole tree constructed so far is at least k.*

Proof. This proof is not trivial because among these subtrees those with ancestry relationship may contain a same expanded cross node.

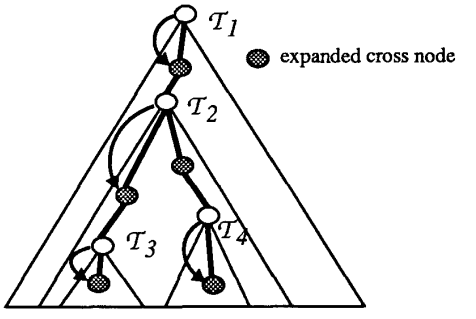

Figure 7: Expanded cross nodes corresponding to PA-independent subtrees.

In this proof, we will prune the k PA-independent subtrees one by one under the restriction that the subtree being pruned contains no other subtrees which have not been pruned yet. (For the example illustrated in Figure 7, we can prune the subtrees in the order: $\mathcal{T}_4$, $\mathcal{T}_3$, $\mathcal{T}_2$, and $\mathcal{T}_1$.) For this proof, it suffices to prove that each pruned subtree has at least one expanded cross node.

Initially, the first pruned subtree obviously has at least one expanded cross node by the assumption of the lemma. As mentioned in Section 5.1, for any two PA-independent subtrees $\mathcal{T}$ and $\mathcal{T}'$ rooted at nodes r and r' respectively, if r is an ancestor of r', there must exist at least one expanded cross node on the path from r (inclusive) to the parent (inclusive) of r' due to processor independence. Therefore, if we prune $\mathcal{T}'$ at r', $\mathcal{T}$ still has at least one expanded cross node. Hence, after we prune each subtree under the above restriction, each of the remaining subtrees will still

have at least one expanded cross node. This implies that the next pruned subtree also has at least one expanded cross node. So, each pruned subtree has at least one expanded cross node. □

Lemma 4 *At some time, if there are k PA-independent HFDC-subtrees and fewer than k expanded cross nodes, there exists an HFD-subtree.* □

Proof. Assume that there exists no HFD-subtree. Thus, each of these PA-independent HFDC-subtrees has at least one expanded cross node according to the definition of HFDC-subtree. By Lemma 3, there are at least k expanded cross nodes. This is contradictory to the assumption of the lemma. □

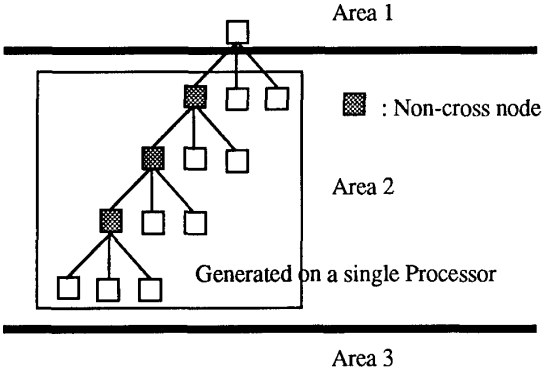

Figure 8: In stage 1, any non-cross node's ancestors in area 2 must have been generated on the same processor.

Lemma 5 *In stage 1, if a processor has expanded h' non-cross nodes, then there are at least κ PA-independent HFDC-subtrees on the processor.*

Proof. As mentioned in Section 5.2, each subtree rooted in area 1 or 2 is always an HFDC-subtree in stage 1. Thus it suffices to prove that at least κ nodes with ancestry independence in areas 1 and 2 will be generated on the processor after h' non-cross nodes have been expanded. By rules R1-R3, for any non-cross node, all of its ancestors in area 2 (with $h'+1$ levels) must be non-cross nodes as shown in Figure 8. So, all the nodes generated by the first h' non-cross nodes must be in areas 1 and 2. Since each of the h' non-cross nodes will generate d children and can remove at most one ancestor, these non-cross nodes will, in total, generate at least $(d-1)h'(=\kappa)$ nodes with ancestry independence. □

Lemma 6 *At any time in stage 1 or 2, including time T_1 or T_2, the tree satisfies properties I1-I4 of Lemma 1.*

Proof. It is obvious from rules R1-R7 that I2 and I3 are satisfied. In addition, it is also obvious that I1 holds before condition C3 or C5 becomes true. Consider the first time step when at least $N-(pd+h+d)$ nodes have been generated (i.e., condition C3 or C5 holds). Since the tree has no more than $N-(pd+h+d)$ nodes in the previous time step and since at most pd nodes will be generated in each time step, there are at most $N-h-d$ nodes in the current time step. In the rest of this proof, we will show that I4 always holds (i.e., there always exists an HFD subtree) in each stage.

In stage 1, all the nodes in area 1 will generate d nodes by rule R1. So, before all the nodes in area 1 have been expanded, there must exist one frontier node in area 1, of which the subtree (with only one node) is an HFD-subtree. After all the nodes in area 1 are expanded, there are at least $d^{\lceil \log_d pdh \rceil} \geq pdh \geq C'$ subtrees rooted at the top level of area 2. Obviously, these subtrees are PA-independent. They are also HFDC-subtrees because each subtree rooted in area 1 or 2 in stage 1 is always an HFDC-subtree as described in Section 5.2. Since the number of expanded cross nodes is always less than C' (due to condition C2), there has always been an HFD-subtree up to time T_1 by Lemma 4. Thus, we can conclude that there always exists an HFD-subtree in stage 1.

In stage 2, initially, there are at least C' PA-independent HFDC-subtrees in Γ (property B1). These subtrees will continue to be HFDC-subtrees in this stage as described in Section 5.2. In stage 2, due to condition C4 the number of expanded cross nodes is always less than C'; so, there always exists an HFD-subtree by Lemma 4. □

Acknowledgement

Comments from Robert Cohn, Routo Terada, and Shang-Hwa Teng are appreciated.

References

[1] E. A. Arnould, F. J. Bitz, E. C. Cooper, H. T. Kung, R. D. Sansom, and P. A. Steenkiste. The design of Nectar: a network backplane for heterogeneous multicomputers. In *Third Intern. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS III)*, Boston, Massachusetts, April 1989.

[2] H.E. Bal. *The shared data-object model as a paradigm for programming distributed systems.* PhD thesis, Vrije Universiteit, Amsterdam, Netherlands, 1989.

[3] C. Ferguson and R.E. Korf. Distributed tree search and its application to alpha-beta prunning. In *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI 1988)*, pages 128–132, Saint Paul, August 1988.

[4] R. Finkel and U. Manber. DIB – a Distributed Implementation of Backtracking. *ACM Transactions on Programming Languages and Systems*, 9(2):235–256, April 1987.

[5] R.M. Haralick and G.L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial intelligence*, 14(3):263–313, October 1980.

[6] C.A.R. Hoare. Quicksort. *Computer Journal*, 5:10–15, 1962.

[7] D.N. Jayasimha. *Communication and Synchronization in Parallel Computation.* PhD thesis, Dept of CS, University of Illinois at Urbana-Champaign, September 1988.

[8] D.N. Jayasimha and M.C. Loui. The communication complexity of parallel algorithms. Technical Report CSRD 629, University of Illinois at Urbana-Champaign, 1986.

[9] V. Kumar and V. N. Rao. Parallel depth-first search, part I: implementation. *International Journal of Parallel Programming*, 16(6):479–499, 1987.

[10] J.M. Ortega and R.G. Voigt. Solution of partial differential equations on vector and parallel computers. *SIAM Review*, 27(2):149–240, 1985.

[11] C.H. Papadimitriou and J.D. Ullman. A communication-time tradeoff. *SIAM J. Comput.*, 16(4):639–646, August 1987.

[12] F.P. Preparata and M.I. Shamos. *Computational Geometry: an introduction.* Springer-Verlag, New York, 1985.

[13] H. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image processing, and GIS.* Addison-Wesley, Reading, MA., 1990.

[14] W. Shu and L. V. Kale. A dynamic scheduling strategy for Chare-Kernel system. In *Proceedings of Supercomputing '89*, pages 389–398, New York, NY, November 1989.

[15] I.-C. Wu. Efficient parallel divide-and-conquer for a class of interconnection topologies. To appear in *the Second Annual International Symposium on Algorithms*, Taipei, December 1991.

[16] Y. Zhang. *Parallel Algorithms for Combinatorial search problems.* PhD thesis, U.C. Berkeley, November 1989.