Scheduling Tree Dags on Parallel Architectures

Konstantinos Kalpakis[1] and Yaacov Yesha[1,2,3]

(Please send all correspondence to: Yaacov Yesha, Computer Science Department, University of Maryland Baltimore County, 5401 Wilkens Avenue, Baltimore, MD 21228-5398, U.S.A.; yayesha@cs.umbc.edu)

Abstract

We provide optimal within a constant explicit upper bounds on the makespan of schedules for tree structured programs on mesh arrays of processors, and provide polynomial time algorithms to find schedules with makespan matching these bounds. In particular, we show how to find, in polynomial time, a (non–preemptive) schedule for a binary tree dag with n unit execution time tasks and height h on a d–dimensional mesh array with m processors and links of unit bandwidth and unit propagation delay whose makespan is $O(n/m + n^{1/(d+1)} + h)$, *i.e.* optimal within a constant factor. Further, we extend these schedules to bounded degree forest dags with arbitrary positive integer execution time tasks and to meshes when the propagation delay of all the links is an arbitrary positive integer. Thus, we provide a polynomial time approximation algorithm for an NP-hard problem, with performance ratio that is a constant.

We also show how to schedule tree dags on any parallel architecture that satisfies certain natural, not very restrictive, conditions that are satisfied by most parallel architectures used in practice. Let ϵ be a fixed positive real number. We provide polynomial time computable schedules for binary tree dags with n unit execution time tasks and height $h \notin (g(n)n^{-\epsilon}, g(n)\log n)$ on any parallel architecture satisfying those conditions, with unit bandwidth and unit propagation delay links, with optimal up to a constant makespan $O(g(n) + h)$, where g is a function that depends only on that architecture. The number of processors used is optimal within a constant factor if $h \leq g(n)n^{-\epsilon}$, and is optimal within an $O(\log n)$ factor if $h \geq g(n)\log n$. As an example, for hypercube and complete binary tree architectures, we achieve optimal within a constant makespan $O(h)$ when $h = \Omega(\log^2 n)$, using an optimal within an $O(\log n)$ factor number of processors. Further, we extend these schedules to the case of bounded degree forest dags with tasks of arbitrary positive integer execution times and architectures when the propagation delay of all the links is a given arbitrary positive integer.

Keywords: multiprocessing, parallel computation, parallel architectures, communication delay, scheduling, tree dags, linear array, mesh array, tree decomposition.

[1]Computer Science Department, University of Maryland Baltimore County, 5401 Wilkens Avenue, Baltimore, MD 21228-5398, U.S.A. E-mail: kalpakis@cs.umbc.edu and yayesha@cs.umbc.edu

[2]Supported in part by the National Science Foundation under grant number CCR-9106062.

[3]Also, supported in part by the University of Maryland at College Park, Institute for Advanced Computer Studies.

1 Introduction

An important consideration in mapping the computational structure of a program onto a multiprocessor system is to keep a good balance between communication overhead and computation time. Moreover, in most multiprocessor systems not every two processors are connected directly by a communication link. A program is represented by a directed acyclic graph (dag). Nodes represent tasks with positive integer execution (computation) times. Edges represent precedence constraints and functional dependencies among tasks. A parallel machine is modeled by an undirected connected graph. Nodes represent identical processors and edges represent communication links. Each processor has its own local memory and is capable of executing any task. Links have propagation delay and constant bandwidth. The propagation delay of all the links is an arbitrary positive integer. Throughout this paper, we assume that each link has unit bandwidth, and unless we say otherwise, all links have one unit propagation delay.

In this paper, we consider the problem of finding efficient schedules for tree structured programs on parallel machines, and in particular on mesh arrays of processors.

Papadimitriou and Ullman [11], Papadimitriou and Yannakakis [12], Jung, Kirousis, and Spirakis [7], and Aggarwal, Chandra, and Snir [1] study the problem of finding efficient methods to execute given programs on parallel machines. Our model differs from the model of Papadimitriou and Yannakakis [12] since in their model there is no notion of limited bandwidth and all communication steps take the same time. Further, our model differs from the model of Aggarwal, Chandra, and Snir [1] since in their model pipelining is not allowed and all communication steps take the same time.

Ghosal, Mukherjee, Thurimella, and Yesha [5] give a polynomial time algorithm to find a schedule for a bounded degree tree dag with n unit execution time tasks and height h on a d-dimensional mesh with $O(\min\{n^{1/(d+1)}, n/h\})$ processors achieving makespan $O(\max\{n^{1/(d+1)}, h\} \log n)$ (optimal and processors–optimal within an $O(\log n)$ factor). (Throughout this paper, log denotes the base 2 logarithm. The makespan of a schedule equals the time to execute that schedule.) Further, Ghosal *et al* [6] extend their schedules to bounded degree tree dags with tasks of arbitrary positive integer execution times. Kalpakis and Yesha [8] give, for any fixed positive real ϵ, a polynomial time algorithm that finds a schedule for a bounded degree tree dag with n unit execution time tasks and height $h \notin (n^{1/2-\epsilon}, n^{1/2} \log n)$ on a linear array achieving optimal within a constant makespan $O(n^{1/2} + h)$, while the time–processors product is optimal within a constant when $h \leq n^{1/2-\epsilon}$ and is optimal within $O(\log n)$ when $h \geq n^{1/2} \log n$. They extend those schedules to d-dimensional meshes when the tree dags have height $h \geq dn^{1/(d+1)} \log n$, achieving $O(h)$ makespan. Kalpakis and Yesha [8] also show that the makespan of an optimal schedule for bounded degree tree dags with n unit time tasks and height h for the model of [12] with parameter τ is $O(\tau \log(n/\tau)/\log(\tau/h))$. Further, they prove that, for bounded degree tree dags with unit time tasks, the linear array is strictly more powerful than the architecture independent model of Papadimitriou and

Yannakakis [12] with interprocessor communication delay equal to the number of processors minus 1. They show that there exist binary tree dags with n unit time tasks and height $o(n^{1/2})$ whose optimal schedule for the model of Papadimitriou and Yannakakis [12] has makespan $\Omega(\sqrt{n \log n}/\log\log n)$ and they provide schedules for those tree dags on linear arrays with optimal within a constant makespan $O(n^{1/2})$.

We provide optimal within a constant explicit upper bounds on the makespan of (non-preemptive) schedules for tree dags on mesh arrays of processors, and polynomial time algorithms to find schedules with makespan matching these bounds. In particular, we show how to find, in polynomial time, a schedule for a binary tree dag with n unit execution time tasks and height h on a d–dimensional (square) mesh array with m processors, with unit bandwidth and unit propagation delay links, whose makespan is $O(n/m + n^{1/(d+1)} + h)$, which is optimal within a constant factor. In constructing these schedules, we develop a new method, that we call path–centroid decomposition, for decomposing a tree into subtrees of balanced size. Our schedules improve by an $O(\log n)$ factor upon the makespan and time–processors product of the schedules in Ghosal *et al* [5]. Further, we show how to extend these schedules to schedules for bounded degree forest dags with arbitrary positive integer execution time tasks on meshes when the propagation delay of all the links is an arbitrary positive integer. Thus, we provide a polynomial time approximation algorithm with performance ratio that is a constant for an NP-hard problem (NP-hardness is established by a straightforward reduction from MULTIPROCESSOR SCHEDULING, which was proven NP-hard in Garey and Johnson [4]).

We also provide optimal within a constant explicit upper bounds on the makespan of schedules for tree dags on any parallel architecture satisfying certain reasonable conditions, and provide polynomial time algorithms to find schedules with makespan matching these bounds. A parallel architecture $\mathcal{M}$ is modeled by a family of undirected connected graphs. For any architecture $\mathcal{M}$, we define three integer functions S, f, and g as follows. For any positive integer m, $S(m)$ is equal to the number of processors in the smallest, with respect to number of processors, machine of architecture $\mathcal{M}$ with $\geq m$ processors. Function f is defined so that $f(m)$ equals the diameter of a machine with $S(m)$ processors, and function g is defined so that $g(n) = \min\{\tau : \tau \geq f(\lceil n/\tau \rceil), \text{ integer } \tau \geq 1\}$. We say that architecture $\mathcal{M}$ is *admissible* if (i) f is non-decreasing, (ii) $S(m) = O(m)$, and (iii) for any k, every k-node subgraph of a machine of architecture $\mathcal{M}$ has diameter $\Omega(f(k))$. Most parallel architectures that are used in practice are admissible, such as meshes, hypercubes, trees, butterflies etc. We provide polynomial time computable schedules for bounded degree tree dags with n unit execution time tasks and height h on any admissible architecture, whose links have unit bandwidth and unit propagation delay, with an optimal within a constant factor makespan $O(g(n) + h)$, when $h \notin (g(n)n^{-\epsilon}, g(n)\log n)$. The number of processors used is optimal within a constant factor when $h \leq g(n)n^{-\epsilon}$, and is optimal within an $O(\log n)$ factor when $h \geq g(n)\log n$. For example, since for hypercube and complete binary tree architectures $g(n) = \Theta(\log n)$, using our schedules for admissible architectures above, we can find, in polynomial time, a schedule for a bounded degree tree dag T with n unit time tasks and

height $h = \Omega(\log^2 n)$ on a hypercube or a complete binary tree architecture with an optimal within a constant factor makespan $O(h)$ and an optimal within a $O(\log n)$ factor number of processors. Note that, for hypercubes, one can obtain, via an embedding result in Bhatt *et al* [2], a schedule for a binary tree dag with makespan $O(h)$, but with much more processors, namely n. We extend our schedules for admissible architectures to the case of bounded degree forest dags with tasks of arbitrary positive integer execution times and to machines of architecture $\mathcal{M}$ when the propagation delay for all the links is a given arbitrary positive integer.

Our schedules for admissible architectures use a tree decomposition for tree dags induced by the method of Papadimitriou and Yannakakis [12], and a simulation technique that we provide for their model by a parallel architecture with diameter no more than the interprocessor communication delay in [12]. It is important to note that, because in a parallel architecture links have constant bandwidth, because an efficient schedule for the model of Papadimitriou and Yannakakis [12] may use too many processors, and because the model of [12] over-estimates the interprocessor communication delay by taking it always to be the machine diameter, the schedule provided by the method of Papadimitriou and Yannakakis [12] does not necessarily provide an efficient schedule for that architecture. Careful placement of tasks, in order to limit link congestion, while making sure that the number of processors used is not too large, is required.

The rest of the paper is organized as follows. In Section 2 we include the needed definitions. In Section 3 we describe the path–centroid decomposition. In Section 4 we present our polynomial time computable optimal within a constant schedules for tree dags on 2–dimensional meshes, and in Section 5 we generalize those schedules to higher dimensional meshes. In Section 6 we present our schedules for tree dags on admissible parallel architectures.

2 Preliminaries

A *bounded degree tree dag* T is a rooted directed bounded degree tree, where the edges are directed towards the root of the tree. (The degree of a node of T equals the number of its predecessors and the degree of T is the maximum of the degrees of its nodes.) Nodes represent computational tasks and edges represent both precedence constraints and functional dependencies among tasks. Each task u has a positive integer execution (computation) time $w(u)$. A bounded degree forest dag F is a collection of bounded degree tree dags. The height of a forest dag is the maximum of the heights of its tree dags. For simplicity, we write $v \in F$ or $(u, v) \in F$ whenever v or (u, v) is a node or an edge in F respectively. In a dag F, node v is called *successor* of node u if $(u, v) \in F$, and node v is called a *predecessor* of node u if $(v, u) \in F$. A *leaf node* is a node with no predecessors. We define the *index of the root* of T to be the height of T plus 1, and the *index* of any other node $u \in T$ to be the height of its

successor node in T. Hereafter, unless we state otherwise, we assume binary tree dags with tasks of unit execution times.

Given any two integers $d \geq 0$ and $b \geq 1$, we view a d–dimensional (square) mesh array of processors with edges each of length b (that is, there are b processors on each edge) as an integer lattice in a d–dimensional space $[1, b]^d$. Throughout this paper, we use this view and standard (and intuitive) geometric concepts (such as lines, points, planes, etc) when referring to a d–dimensional mesh. For simplicity, we refer to a 2–dimensional square mesh simply as a mesh.

Tasks are assigned to processors for execution. A task may be assigned to more than one processor, in which case this processor holds a copy of that task. If there is at least one task with more than one copy then we say that we have *recomputation*. Recomputation is unnecessary for forest dags [5], while it is necessary for inverse forest dags [7]. All our schedules have no recomputation. For simplicity, we refer to a copy of a task simply as a task. We say that *a task is ready* if the values of all its predecessors are available to it. Processors perform computation according to the following eight rules:

(1) Computation is synchronized.

(2) Execution of tasks is non-preemptive.

(3) A non-leaf task can not be executed before it becomes ready. All leaf tasks are ready.

(4) Each processor can execute in $w(u)$ time units a copy of a task u that is assigned to it.

(5) At each time unit at most one value can be sent over a link.

(6) A value sent over a link arrives at the other end of that link after a number of time units equal to the propagation delay of that link.

(7) After a copy of a task is executed, its value is available to the processor to which it is assigned.

(8) If a value is transmitted by a link to a processor then it becomes available to that processor.

The *makespan* $T_{\max}$ of a schedule is the number of time units that pass until all copies of each task are executed. Given a dag and a parallel architecture, a schedule is called *optimal* if its makespan $T_{\max}$ is minimum among all possible schedules for that dag on that architecture. A schedule is called *processors–optimal* with respect to a given time t if the number of processors used is minimum among all schedules for that dag on that architecture whose makespan is t.

Given a bounded degree forest dag F and a parallel architecture, our objective is to find a schedule for F on that architecture, with the following two properties: (i) Its makespan $T_{\max}$

is optimal or close to optimal. (ii) The number of processors used is close to the minimum number of processors required to achieve time $T_{\max}$.

3 Path–Centroid Decomposition of a Tree Dag

Let T be a binary degree tree dag with n nodes and height h. It is well known that, by removing an appropriate edge from T, we can partition T into two subtrees each with no more than $\lceil 2n/3 \rceil$ and no less than $\lfloor n/3 \rfloor$ nodes. To find such an edge proceed as follows. Find a path from the root of T to a node u of T such that the subtree that is rooted at u has between $\lfloor n/3 \rfloor$ and $\lceil 2n/3 \rceil$ nodes. The required edge is the edge on that path that is incident to u. This method is known as the *edge–centroid decomposition method*. Given a positive integer β, we can partition T, using this method recursively, into $\leq \lceil 3n/\beta \rceil$ subtrees such that each subtree has no less than $\lfloor \beta/3 \rfloor$ and no more than β nodes [3, 10]. To find such a decomposition of T, we do the following. Remove from T the edge found by applying the edge–centroid decomposition method to T, and recursively decompose each subtree in the resulting forest that has more than β nodes. Such a decomposition of T can always be computed in polynomial time.

Another way to decompose T is to partition it into a set of paths as follows. Take a directed path from a leaf of T to its root, remove that path from T, and recursively decompose each tree in the resulting forest. The set of all such paths forms a partition of T. The number of paths in that partition equals the number of leaves of T. We call such a partition of T *a path decomposition of* T.

We develop yet another way to decompose a tree into subtrees. Given a positive integer $\beta \leq n$, we want to find a decomposition of T into subtrees such that each subtree T_i in that decomposition will satisfy the following two properties:

Property 1: T_i has no more than β nodes.

Property 2: all the nodes of T_i, that have a predecessor in T that is not in T_i, are on a single path from a leaf of T_i to the root of T_i.

Also, we require that such a decomposition of T satisfies the following property:

Property 3: there are at most $2\lceil 3n/\beta \rceil$ subtrees in that decomposition of T.

We find such a decomposition of T by combining the edge–centroid decomposition and the path decomposition methods. We call the resulting method the *path–centroid decomposition method*.

Let β be a positive integer $\leq n$. Using the edge–centroid decomposition method recursively, first decompose T into $\Theta(n/\beta)$ subtrees $R_1, R_2, \ldots, R_{k_\beta}$ such that each subtree has $\geq \lfloor \beta/3 \rfloor$ and $\leq \beta$ nodes. Each subtree R_i satisfies the first property above, but it may fail to satisfy the second property.

We further decompose each subtree R_i that fails to satisfy the second property, into subtrees so that both properties are satisfied. Let R'_i be the subtree of R_i that consists of all the nodes of R_i lying on a directed path from any node of R_i, with a predecessor in T that is not in R_i, to the root of R_i. Let $\pi_1, \pi_2, \ldots, \pi_{k'_i}$ be a path decomposition of R'_i. Observe that the successor of the root of each subtree in the forest $R_i - R'_i$ is in a unique path in that path decomposition of R'_i. Let $R_{i,j}$ be the subtree of R_i that consists of π_j and those subtrees in the forest $R_i - R'_i$ whose roots have their successors on π_j, $j = 1, 2, \ldots, k'_i$. Observe that, for each subtree $R_{i,j}$, all the nodes of $R_{i,j}$ that have a predecessor that is not in $R_{i,j}$ are lying on a single path (in $R_{i,j}$) to the root of $R_{i,j}$. Further, the subtrees $R_{i,j}$, $j = 1, 2, \ldots, k'_i$, form a partition of R_i. Consequently, each such subtree $R_{i,j}$ satisfies both properties above. Finally, for any R_i that was not further decomposed, let $R_{i,1} = R_i$. The path–centroid decomposition $T_1, T_2, T_3, \ldots$ of T consists of all the $R_{i,j}$'s above. Clearly, this decomposition can be computed in polynomial time.

Lemma 1 *Let T be a binary tree dag with n nodes and let β be a positive integer $\leq n$. Then, using the path–centroid decomposition method, we can decompose T into no more than $2\lceil 3n/\beta \rceil$ subtrees $T_1, T_2, T_3, \ldots$ so that each subtree T_i has $\leq \beta$ nodes and all nodes of T_i with a predecessor in another subtree are lying on a single path (in T_i) to the root of T_i. Further, this decomposition is polynomial time computable.*

Proof: Consider the method given above for finding a path–centroid decomposition of T. Since each subtree T_k is a subtree $R_{i,j}$, for some positive integers i, j, their properties follow from the discussion above. We only need to find an upper bound on the number of subtrees in that path–centroid decomposition of T. Since we first decompose T using the edge–centroid decomposition method, it follows that the number k_β of subtrees R_i in this decomposition is $\lceil n/\beta \rceil \leq k_\beta \leq \lceil 3n/\beta \rceil$. In addition, if R_i is further decomposed as above (because it violates the second property), then the number k'_i of subtrees $R_{i,j}$ in that decomposition of R_i is no more than the number of subtrees R_l that have the successor of their roots in R_i. Therefore, the total number of subtrees in this path–centroid decomposition of T is no more than $2k_\beta \leq 2\lceil 3n/\beta \rceil$. ■

Suppose now that we are given a path–centroid decomposition $T_1, T_2, T_3, \ldots$ of T. For each subtree T_i in that decomposition we define a path π_i in T_i, which we call the *basic path* that corresponds to T_i. If T_i has a node whose predecessor(s) in T is not in T_i, then π_i is the single path in T_i from that node to the root of T_i. Otherwise, π_i consists of the root of T_i only.

Moreover, given that path–centroid decomposition of T, we construct, by collapsing each subtree into a single supernode, a compressed tree T_c as follows. For each subtree T_i

we have a supernode v in T_c, *i.e.* each supernode represents a subtree in that decomposition of T. There is an edge in T_c from $u \in T_c$ to $v \in T_c$ if the successor node of the root of the subtree represented by u is in the subtree represented by v. We call T_c the *path–centroid compressed tree of* T associated with that path–centroid decomposition of T. The level of a supernode of T_c is equal to its distance (in T_c) from the root of T_c. Define the *level of a subtree* T_i in that decomposition of T to be the level of the supernode of T_c representing T_i.

Perform a breadth-first-search of T_c, viewing it as an undirected rooted tree, starting from its root and number its supernodes with consecutive positive integers, with the root of T_c numbered 1, according to the order in which they are visited. We call the number assigned to each supernode of T_c, in that manner, the *BFS number* of that supernode. The BFS number of a subtree T_i equals the BFS number of the supernode representing T_i.

4 Scheduling Tree Dags on Meshes

We provide a polynomial time algorithm for scheduling tree dags on meshes with optimal within a constant makespan and number of processors. Intuitively, our algorithm is based on assigning basic paths and/or subtrees resulting from a path–centroid decomposition of a tree T or a subtree of T to processors on the leftmost vertical line or on a horizontal line of a mesh. Tasks are executed in a greedy manner as soon as they become ready. Values are routed over suitable selected shortest paths between their source and destination processors. Link contention is resolved according to a first–needed–first–routed policy.

4.1 The Schedule

Let T be a binary tree dag with n unit execution time tasks and height h. We show how to schedule T on a 2-dimensional square mesh with $b = 12B + 1$ processors on each line, where B is a given positive integer $\leq n$. Let $P_{i,j}$ be the processor of this mesh that is at the intersection of the ith vertical line from the left and the jth horizontal line from the bottom, $1 \leq i, j \leq b$.

4.1.1 Task Assignment

Using the path–centroid decomposition method with parameter $\lceil n/B \rceil$, decompose T into $\leq 2\lceil 3n/\lceil n/B \rceil \rceil \leq 6B$ subtrees each with $\leq \lceil n/B \rceil$ tasks. Let $T_1, T_2, T_3, \ldots$ be the subtrees in this decomposition in BFS order. Let π_i denote the basic path that corresponds to subtree T_i. Then, for each subtree T_i do the following:

Case 1: If T_i has $\leq B$ tasks, assign all the tasks in T_i to processor $P_{1,2i}$.

Case 2: Otherwise, do the following. Using the path–centroid decomposition method with parameter $\lceil n/B^2 \rceil$, decompose T_i into $\leq 6B$ subtrees each with $\leq \lceil n/B^2 \rceil$ tasks. Let $T_{i,1}, T_{i,2}, T_{i,3}, \ldots$ be the subtrees in this decomposition of T_i in BFS order. Assign all the tasks on the basic path π_i to processor $P_{1,2i}$, and for each subtree $T_{i,j}$ assign all the tasks in $T_{i,j} - \pi_i$ to processor $P_{j+1,2i+1}$.

Let $p(v)$ denote the processor of the mesh that has been assigned task v of T.

4.1.2 Task Execution

Processors execute the tasks that have been assigned to them as follows. At any time $t \geq 0$, each processor p selects for execution a task u that has minimum index among all tasks that have been assigned to p and are ready at time t (break ties arbitrarily). Processor p executes task u at time t and u's value is available to p at time $t+1$.

4.1.3 Routing

The routing of values of tasks is done as follows. Consider two tasks u and v of T such that v is the successor of u in T and $p(u) \neq p(v)$. The value of u is routed from $p(u)$ to $p(v)$ using the links on a shortest path from $p(u)$ to $p(v)$, such that the first link on that shortest path is parallel to the basic line that contains $p(v)$. A line of the mesh is a *basic line* if it is an odd–numbered horizontal line or it is the leftmost vertical line of the mesh.

Unfortunately, several values may compete to be routed over the same link at the same time. We use the following link contention resolution strategy. Whenever several values of tasks compete to use the same link at the same time, the value of the task with the lowest index is routed first (break ties arbitrarily).

FIG.1 Fig. 1 illustrates an example of our schedule for a tree dag on a mesh.

4.2 Computing an Upper Bound on the Makespan

To find an upper bound on the makespan of our schedule, we compute certain upper bounds on the time it takes to route values of tasks, by analyzing the contribution of the distance traveled and the link congestion encountered. Using these upper bounds, by induction on the height of each task, we find an upper bound on the completion time of each task. In particular, we prove, by induction on the height of each task, that each task v of T starts execution by time $\mathrm{Comp}(n,B) + 3h(v) + 6\mathrm{Dist}(v) + \delta(v) + \mathrm{Cong}^*(v)$, where $h(v)$ is the height of v. The components $\mathrm{Comp}(n,B)$, $\mathrm{Dist}(v)$, $\delta(v)$, and $\mathrm{Cong}^*(v)$, are formally defined below. Intuitively, $\mathrm{Comp}(n,B)$ is an upper bound on the total amount of computation performed

by each processor, $\text{Cong}^*(v)$ is an upper bound on the cumulative communication delay that is due only to link contention, and $\text{Dist}(v)$ and $\delta(v)$ are both used to bound from above the cumulative communication delay that is due only to distances traveled. The height of a task enters in the above inequality because of precedence constraints among tasks. Then, the inequality above intuitively says that the elapsed time in executing a task is bounded from above, within a constant factor, by the sum of its height, the amount of computation performed by each processor, the delay due to link contention, and the delay due to distances traveled.

Let $t(v)$ denote the time at which processor $p(v)$ starts executing a task v of T. Let $t'(v)$ be the time at which the value of v is available at the processor that has been assigned the successor task of v in T, if v has a successor, and be the time at which v finishes execution otherwise. Since each task takes one time unit for execution, $t'(v) \geq t(v) + 1$. Clearly, the makespan of our schedule for T is equal to the time at which the root of T finishes execution.

Consider the task assignment of our schedule for T on the mesh. We say that two processors are *colinear* if they are both on the same basic line. Let $\text{Comp}(n, B) = \max\{B, \lceil n/B^2 \rceil\}$. Intuitively, $\text{Comp}(n, B)$ is an upper bound on the amount of computation performed by any processor of the mesh. In particular, we prove the following.

Lemma 2 *Each processor is assigned at most* $\text{Comp}(n, B)$ *tasks. Only processors on basic lines may have been assigned tasks. At each time unit* $\geq \text{Comp}(n, B)$*, each processor has at most one ready task. Each processor executes at most one task whose value is needed by some other colinear processor.*

Proof: Clearly, no processor is assigned more than $\text{Comp}(n, B)$ tasks. Let p be a processor that has been assigned at least one task. By inspection of the task assignment method, it follows that p is either on the leftmost vertical line or on an odd-numbered horizontal line. Hence, only processors on basic lines may have been assigned tasks. Moreover, it follows from the task assignment method and Lemma 1 that p has been assigned a basic path, and possible some extra tasks which have all their predecessors also assigned to p.

Since all the tasks that have been assigned to p and are not on the basic path that has been assigned to p can be executed by time $\text{Comp}(n, B)$, and since at most one task on that basic path can be ready at any time after that, it follows that, at any time $\geq \text{Comp}(n, B)$, processor p has at most one ready task.

Consider a task u of T, and let v be its successor. Suppose that $p(u) \neq p(v)$ are colinear. Then, task u is the highest task on the basic path that has been assigned to $p(u)$. Since each processor is assigned at most one basic path, each processor executes at most one task whose value is needed by some other colinear processor. ■

We will need the following definitions for computing an upper bound on the makespan.

For each task u of T, let $d(u,v)$ be equal to the distance between processors $p(u)$ and $p(v)$, where v is the successor of u. Intuitively, $d(u,v)$ is an upper bound on the communication delay, due only to distance traveled, for the value of u to be communicated to $p(v)$.

For each task v of T, let $\delta(v)$ be equal to the distance between processor $p(v)$ and the most distant processor that is colinear with $p(v)$ and has been assigned an ancestor of v. Note that if u is a predecessor of v and $p(u), p(v)$ are colinear processors then $\delta(u)+d(u,v) \leq \delta(v)$. Intuitively, $\delta(v)$ is an upper bound on the communication delay, due only to distance traveled, for the values of all ancestors of v, that have been assigned to processors colinear with $p(v)$, to be communicated to $p(v)$.

For each task v of T, let Dist(v) be equal to 0 if processor $p(v)$ is on a horizontal basic line, and be equal to b if $p(v)$ is on the vertical basic line. Intuitively, Dist(v) is an upper bound on the distance the value of a predecessor of v has to travel in order to reach $p(v)$, when it has been assigned on a basic line along a dimension that is lower than the dimension of the basic line containing $p(v)$. (It is assumed that a horizontal line is a line along dimension 1 and that a vertical line is a line along dimension 2.)

Due to the BFS assignment, the values that compete with the value of a task that is the root of a subtree of level k, for using a link on the vertical basic line, are all values of roots of subtrees of level $k-1$, k, or $k+1$ in the path–centroid decomposition of T. The situation is similar for horizontal basic lines. In order to measure the delay due to link contention, we define below, for each task u of T, the following two quantities Cong(u) and Cong$^*(u)$.

Define $\gamma(T,k)$ to be equal to the number of subtrees in the path–centroid decomposition of T which are at levels $k-1$, k, and $k+1$. Define $\gamma^*(T,k)$ to be equal to $\sum_{j=k}^{h'} \gamma(T,j)$, where h' is the height of the compressed tree that corresponds to the path–centroid decomposition of T. Likewise, define $\gamma(T_i,k)$ to be equal to the number of subtrees in the path–centroid decomposition of T_i which are at levels $k-1$, k, and $k+1$. Define $\gamma^*(T_i,k)$ to be equal to $\sum_{j=k}^{h'_i} \gamma(T_i,j)$, where h'_i is the height of the compressed tree that corresponds to the path–centroid decomposition of T_i.

For each task u of T define Cong(u) and Cong$^*(u)$ as follows. There are two cases to consider. In the first case, u has been assigned to a processor, say $P_{1,2i}$, on the vertical basic line. Task u is contained in the subtree T_i. Define Cong(u) to be equal to $\gamma(T,k)$ and Cong$^*(u)$ to be equal to $\gamma^*(T,k+1)$, where k is the level of T_i in the path–centroid decomposition of T. In the second case, u has been assigned to a processor, say $P_{j+1,2i+1}$, on a horizontal basic line. Task u is contained in the subtree $T_{i,j}$. Define Cong(u) to be equal to $\gamma(T_i,k)$ and Cong$^*(u)$ to be equal to $\gamma^*(T_i,k+1)$, where k is the level of $T_{i,j}$ in the path–centroid decomposition of T_i. Observe that if a task u and its successor v in T have been assigned to different colinear processors then Cong$^*(u)$ + Cong(u) = Cong$^*(v)$, since the levels of the subtrees that correspond to u and v differ by one. Intuitively, Cong(u) is an upper bound on the communication delay, due only to link contention, for the value of

u to be communicated to $p(v)$ given that v is the successor of u and that $p(u)$ and $p(v)$ are different colinear processors. Further, $\text{Cong}^*(v)$ is an upper bound on the communication delay, due only to link contention, for the values of all ancestors of v that have been assigned to processors colinear with $p(v)$, to be communicated to $p(v)$.

The following lemma summarizes the properties of $\text{Dist}(u)$, $d(u,v)$, $\delta(u)$, $\text{Cong}(u)$, and $\text{Cong}^*(u)$ that we use later to compute an upper bound on the makespan of our schedule.

Lemma 3 *Let u be a task of T and let v be its successor in T. Then, the following are true:*

- **(a)** *If tasks u and v have been assigned to the same processor, then* $\text{Dist}(u) = \text{Dist}(v)$, $\delta(u) = \delta(v)$, *and* $\text{Cong}^*(u) = \text{Cong}^*(v)$.
- **(b)** *If tasks u and v have been assigned to different colinear processors then* $\text{Dist}(u) = \text{Dist}(v)$, $\delta(u) + d(u,v) \leq \delta(v)$, *and* $\text{Cong}^*(u) + \text{Cong}(u) = \text{Cong}^*(v)$.
- **(c)** *If u and v have been assigned to non-colinear processors then* $\text{Dist}(u) + b \leq \text{Dist}(v)$.
- **(d)** *For any task v of T,* $0 \leq \delta(v) \leq b$, $0 \leq \text{Cong}^*(v) \leq 3b$, *and* $0 \leq \text{Dist}(v) \leq b$.

Proof: Follows from the definitions of the quantities involved. ■

We state now the main theorem of this section.

Theorem 1 *Let T be a binary tree dag with n unit execution time tasks and height h. Given a mesh with m processors, we can find, in polynomial time, a schedule for T on the given mesh with optimal within a constant factor makespan $O(n/m + n^{1/3} + h)$.*

Proof: Let u and v be two tasks of T such that v is the successor of u in T. Recall that $t(u)$ denotes the time at which u starts execution at $p(u)$, and $t'(u)$ denote the time at which u's value is available to its successor. Obviously, if $p(u) = p(v)$ then $t'(u) = t(u) + 1$. In Lemma 4, we show that if $p(v)$ and $p(u)$ are different colinear processors then $t'(u) \leq t(u) + d(u,v) + \text{Cong}(u)$. In Lemma 5, we show that if $p(v)$ and $p(u)$ are non–colinear then, $t'(u) \leq \max\{t(u) + b + 3, t(v') + 3\}$ if task v has a predecessor v' on the basic path assigned to $p(v)$, and $t'(u) \leq t(u) + b + 3$ otherwise. Then, in Lemma 6, using the three inequalities above and Lemmas 2 and 3, we show, by induction on the height of each task, that $t(v) \leq \text{Comp}(n,B) + 3h(v) + 6\text{Dist}(v) + \delta(v) + \text{Cong}^*(v)$. From part (d) of Lemma 3, and since $\text{Comp}(n,B) = \max\{B, \lceil n/B^2 \rceil\}$ and $b = 12B + 1$, it follows that the makespan of our schedule for T is $\leq \max\{B, \lceil n/B^2 \rceil\} + 120B + 3h + 11$ and it uses $(12B+1)^2$ processors. By taking $B = \min\{\lceil n^{1/3} \rceil, \lceil (n/h)^{1/2} \rceil, \max\{1, \lfloor (m^{1/2} - 1)/12 \rfloor\}\}$, it follows that we can find, in polynomial time, a schedule for T on a mesh with m processors and makespan $O(n/m + n^{1/3} + h)$.

We note here that $\Omega(n/m + n^{1/3} + h)$ is a lower bound on the makespan of any schedule for T on a 2-dimensional mesh with m processors (for details see Ghosal *et al* [5]). (This lower bound is basically the maximum of the number of tasks assigned to each processor, the height of the tree, and half the diameter of the mesh, *i.e.* $T_{\max} = \Omega(\max\{n/m, m^{1/2}, h\}) \geq \Omega(n/m + n^{1/3} + h)$.) Thus, our schedule for T is optimal within a constant. ■

In the remainder of this section, we prove Lemmas 4, 5, and 6. At the end of the section we explain how to extend our schedules to bounded degree forest dags with arbitrary positive integer execution times and to meshes when the propagation delay of all the links is an arbitrary positive integer.

Lemma 4 *Let u and v be two tasks of T such that v is the successor of u and processors $p(u) \neq p(v)$ are colinear. Then, the value of task u is available to processor $p(v)$ at time $t'(u) \leq t(u) + d(u, v) + \mathrm{Cong}(u)$.*

Proof: If $p(u)$ and $p(v)$ are on the vertical basic line, then u is the root of a subtree in the path–centroid decomposition of T. If $p(u)$ and $p(v)$ are on the $(2i+1)$th horizontal basic line, for some i, then u is the root of a subtree in the path–centroid decomposition of T_i.

Suppose that both $p(u)$ and $p(v)$ are on the vertical basic line. The case where both are on a horizontal basic line is similar. Let k be the level of the subtree in the path–centroid decomposition of T that has u as its root.

The time to communicate the value of task u from $p(u)$ to $p(v)$ is bounded as follows. The distance the value of task u travels is $d(u, v)$. Due to the BFS assignment, the links on the shortest path from $p(u)$ to $p(v)$ can only be used for routing the values of the roots of the subtrees in the path–centroid decomposition of T whose levels are $k-1$, k, or $k+1$. Thus, the value of u can be delayed, due to link contention, by no more than $\mathrm{Cong}(u) - 1$ other values. Consequently, the time to communicate the value of u from $p(u)$ to $p(v)$ is $\leq d(u, v) + \mathrm{Cong}(u) - 1$. Since the value of u is available to $p(u)$ at time $t(u) + 1$, the lemma follows. ■

Lemma 5 *Let u and v be two tasks of T such that v is the successor of u and processors $p(u) \neq p(v)$ are non–colinear. Let π be the basic path of T assigned to $p(v)$. Then, the value of task u is available to $p(v)$ at time $t'(u) \leq \max\{t(u) + b + 3, t(v') + 3\}$ if task v has a predecessor v' on π, and at time $t'(u) \leq t(u) + b + 3$ otherwise.*

Proof: In order to find the time by which the value of task u is available to processor $p(v)$, we need to find an upper bound on the distance the value of u has to travel when it is routed from $p(u)$ to $p(v)$. Further, we need to examine the effect of link contention on the time it takes to communicate the value of u to $p(v)$. Note that, by the task assignment method, both tasks are contained in a subtree T_i in the path–centroid decomposition of T, π is the

basic path that corresponds to T_i, and v is on π. In addition, $p(v)$ is processor $P_{1,2i}$ on the vertical basic line, while $p(u)$ is processor $P_{j+1,2i+1}$, $1 \leq j \leq 6B$, on a horizontal basic line.

First, we find a bound on the distance the value of u travels. The value of u is routed from $P_{j+1,2i+1}$ to $P_{j+1,2i}$ using the link that connects them. Then, it is routed from $P_{j+1,2i}$ to $P_{1,2i}$ using the links on the shortest path between these two processors, which are all on the $2i$th horizontal line of the mesh. Let E_u be the set of links used. The distance the value of u travels is equal to $|E_u| = j + 1 \leq b$.

Second, we find the delay the value of u will experience due to link contention. Observe that if in routing the value of a task u' of T a link in E_u is used then task u' is in $T_i - \pi$ and the successor of u' is on the basic path π.

Because of the link contention resolution strategy, the value of u can be delayed only by values of tasks whose index is less than or equal to the index of u. Recall that the index of a task is the height of its successor task. Since T is a binary tree, there is at most one more task with index equal $h(v)$ that may use a link in E_u. In addition, the successor of such a task is task v. Hence, the value of u can be delayed by at most one other value of a task with index equal to $h(v)$.

There are two cases to consider. In the first case, the value of u is delayed only by values of tasks with index $h(v)$. Since the value of u is available to $p(u)$ at time $t(u) + 1$, it follows that the value of u is available to $p(v)$ at time $t'(u) \leq t(u) + b + 3$.

In the second case, the value of u is delayed by the value of a task with index less than $h(v)$. Then, v must have a predecessor v' on π. Let u' be the task with index less than $h(v)$ that delayed the value of u last. Since the successor of u' is also assigned to $p(v)$, the value of u' must arrive at $p(v)$ no later than the time at which v' becomes ready, *i.e.*, $t'(u') \leq t(v')$. Since the value of u may be delayed by at most one other value of a task with index equal to $h(v)$, and since u' is the task with index less than $h(v)$ whose value delayed the value of u last, the value of u arrives at processor $p(v)$ by time $t'(u') + 1 + 1 \leq t(v') + 3$. The lemma now follows. ■

Lemma 6 *Each task v of T is executed at time*

$$t(v) \leq \text{Comp}(n, B) + 3h(v) + 6\text{Dist}(v) + \delta(v) + \text{Cong}^*(v). \quad (1)$$

Proof: We prove this claim by induction on the height of each task.

Basis: Let v be a task of height $h(v) = 0$. Task v is ready at time 0. By Lemma 2 each processor is assigned $\leq \text{Comp}(n, B)$ tasks. Thus, $p(v)$ will start executing task v by time $\text{Comp}(n, B)$. Because all other terms on the right hand-side of (1) are non–negative, the claim is true for v.

Inductive hypothesis: Suppose that the claim is true for each task of height $< j$.

Inductive step: Let v be a task of height $h(v) = j$. Let π be the basic path that has been assigned to processor $p(v)$. If v is not on π then, since all predecessors of v are also assigned to $p(v)$ and $p(v)$ has been assigned $\leq \text{Comp}(n, B)$ tasks, the claim is true for v. Thus, suppose that v is on π. Task v becomes ready when the values of all its predecessors are available to $p(v)$. Let u be a predecessor of v in T. Since $h(u) \leq h(v) - 1 < j$, by the inductive hypothesis for u, processor $p(u)$ starts executing task u at time $t(u) \leq \text{Comp}(n, B) + 3h(u) + 6\text{Dist}(u) + \delta(u) + \text{Cong}^*(u)$. We show that the value of u is available to $p(v)$ at time

$$t'(u) \leq \text{Comp}(n, B) + 3h(v) + 6\text{Dist}(v) + \delta(v) + \text{Cong}^*(v). \tag{2}$$

There are three cases to consider.

Case 1: processors $p(u)$ and $p(v)$ are identical. The value of u is available to $p(v)$ at time $t'(u) = t(u) + 1$. Then, (2) follows from the inductive hypothesis for u and part (a) of Lemma 3.

Case 2: $p(u)$ and $p(v)$ are different colinear processors. By Lemma 4, the value of u becomes available to $p(v)$ at time $t'(u) \leq t(u) + d(u, v) + \text{Cong}(u)$. Inequality (2) follows from the inductive hypothesis for u and part (b) of Lemma 3.

Case 3: $p(u)$ and $p(v)$ are non–colinear. Processor $p(u)$ is on a horizontal basic line and $p(v)$ is on the vertical basic line. By part (c) of Lemma 3, $\text{Dist}(u) + b \leq \text{Dist}(v)$. There are two sub-cases to consider.

Case 3.1: v does not have a predecessor on π. By Lemma 5, $t'(u) \leq t(u) + b + 3$. Since $h(u) \leq h(v) - 1$, (2) follows from the inductive hypothesis for u and parts (c) and (d) of Lemma 3.

Case 3.2: v has a predecessor v' on π. By Lemma 5, the time at which the value of u is available to $p(v)$ is $t'(u) \leq \max\{t(u) + b + 3, t(v') + 3\}$. Since v' and v have been assigned to the same processor and $h(v') \leq h(v) - 1$, (2) follows from the inductive hypothesis for v', part (a) of Lemma 3, and the analysis in case 3.1.

Consequently, the value of any predecessor u of v is always available to $p(v)$ by time $\text{Comp}(n, B) + 3h(v) + 6\text{Dist}(v) + \delta(v) + \text{Cong}^*(v)$, which implies that task v becomes ready by that time. Since by Lemma 2 each processor has at most one ready task at each time unit $\geq \text{Comp}(n, B)$, inequality (1) is true for each task v of T. ■

Note that our schedule for T on a mesh induces a schedule for T on a linear array with m processors and optimal within a constant makespan $O(n/m + n^{1/2} + h)$.

Next, we extend our schedules for binary tree dags to bounded degree tree dags with tasks of arbitrary positive integer execution times and to meshes when the propagation delay

of all the links is an arbitrary positive integer. To this end, we generalize, in a straightforward manner, the path–centroid decomposition method to bounded degree weighted trees. Then, we can show the following. Suppose that we are given a weighted bounded degree tree dag T with each task of T having an arbitrary positive integer weight (*i.e.* execution time). Suppose also that we are given a mesh with m processors and with links each with r_0 time units propagation delay. Let W be the sum of the weights of all the tasks of T. Let the *weighted height* h_w of T be equal to the maximum sum of the weights of all the tasks of T on a directed path in T, where the maximum is taken over all directed paths in T. Then, we can find, in polynomial time, a schedule for T on the given mesh whose makespan is $O(W/m + (Wr_0^2)^{1/3} + h_w)$, *i.e.* optimal within a constant.

In addition, we can schedule forest dags on meshes. Let F be a binary forest dag with n unit execution time tasks and height h. Let $n_{\max}$ be the maximum number of tasks of any tree in F. Suppose that a mesh with m processors is given. To schedule F do the following. First, combine the small trees in F, by adding $O(n)$ dummy tasks, to trees with $\Theta(n_{\max})$ tasks. Second, schedule each tree, which has $\Theta(n_{max})$ tasks, on disjoint sub–meshes of the given mesh using our schedule for tree dags. The makespan of the resulting schedule is $O(n/m + n_{\max}^{1/3} + h)$, *i.e.* optimal within a constant. This approach generalizes to schedules for bounded degree weighted forest dags on meshes when the propagation delay for all the links is an arbitrary positive integer.

5 Scheduling Tree Dags on Higher Dimensional Meshes

We give a polynomial time algorithm for scheduling tree dags on d–dimensional meshes with optimal within a constant makespan and number of processors. Intuitively, our algorithm is based on assigning basic paths resulting from the path–centroid decomposition of a tree to processors on a line of the d-mesh, and then recursively assigning the forests of tasks on which those paths depend to distinct submeshes perpendicular to that line. The algorithm is a generalization of our scheduling algorithm for tree dags on meshes.

5.1 The Schedule

Let T be a binary tree dag with n unit execution time tasks and height h. Let B be a positive integer $\leq n$, and $b = 12B + 1$. We show how to find a schedule for T on a d–dimensional (square) mesh with $(12B + 1)^d$ processors and $O(n/B^d + dB + h)$ makespan, $d \geq 2$. By choosing appropriate values for B, we obtain the desired schedules.

Let L_d be the line of the d-dimensional mesh that is along dimension d and passes through processor $P_{1,1,\ldots,1}$. Line L_d is a *basic line* along dimension d. Assign the tasks of T to the processors of the d-dimensional mesh recursively as follows.

Case 1: $d = 2$. Use our schedule for meshes in the previous section.

Case 2: $d > 2$. Compute a path–centroid decomposition of T with parameter $\lceil n/B \rceil$. Let $T_1, T_2, T_3, \ldots$ be the subtrees in that decomposition in BFS order. Let π_i be the basic path that corresponds to T_i. For each subtree T_i do the following. If T_i has $\leq B$ tasks assign all the tasks in T_i to the $2i$th processor on L_d. Otherwise, do the following. Assign all the tasks on π_i to the $2i$th processor on L_d. Let $\mathcal{P}_{2i+1}$ be the $(d-1)$–dimensional submesh of the d–dimensional mesh that is perpendicular to L_d at its $(2i+1)$th processor. Let L_{d-1} be the line of $\mathcal{P}_{2i+1}$ that is along dimension $d-1$ and intersects the line L_d. Line L_{d-1} is a *basic line* along dimension $d-1$. Assign, recursively, the tasks in T_i to the processors of the $\mathcal{P}_{2i+1}$ submesh, using line L_{d-1} instead of line L_d. Remove from the processors of $\mathcal{P}_{2i+1}$ any task that is on π_i.

FIG.2 Fig. 2 illustrates an example of the task assignment method.

The processors of the d–dimensional mesh execute the tasks that have been assigned to them following the same regime as for the case of 2–dimensional meshes, *i.e.* the ready task with smallest index is executed first.

The routing of values of tasks is done as follows. Consider two tasks u and v of T such that v is the successor of u in T and such that processors $p(u)$ and $p(v)$ are different. The value of u is routed from $p(u)$ to $p(v)$ using the links on a shortest path in the d–mesh from $p(u)$ to $p(v)$, such that the first link on that shortest path is parallel to the basic line that contains $p(v)$. We use the same link contention resolution policy as in our schedules for 2–dimensional meshes, *i.e.* the value of a task with the smallest index is routed first. Fig. 3
FIG.3 illustrates an example of routing values of tasks in a d–dimensional mesh.

5.2 The Makespan

Using similar arguments as in the case of 2–dimensional meshes we show the following.

Theorem 2 *Suppose that we are given a binary tree dag T with n tasks and height h, and a d–dimensional mesh with m processors. Then, we can find, in polynomial time, a schedule for T on the given mesh with optimal within a constant makespan $O(n/m + n^{1/(d+1)} + h)$.*

Proof sketch: (See [9] for a detailed proof.) Similar to the proof of Theorem 1.

We extend, in a straightforward manner, the definitions of $\mathrm{Comp}(n, B)$, $\mathrm{Dist}(u)$, $d(u, v)$, $\delta(u)$, $\mathrm{Cong}(u)$ and $\mathrm{Cong}^*(u)$ to d–dimensional meshes. For example, $\mathrm{Comp}(n, B) = \max\{B, \lceil n/B^d \rceil\}$, $\mathrm{Dist}(u) = (i-1)b$ for each task u assigned on a basic line along dimension i, and $d(u, v)$ is the distance between tasks assigned to colinear processors. The definitions of $\delta(u)$, $\mathrm{Cong}(u)$, and $\mathrm{Cong}^*(u)$ are similar to the ones for the 2–dimensional meshes.

Then, we extend Lemmas 2–6 to the case of d–dimensional meshes. Lemma 2 holds as is. Parts (a) and (b) of Lemma 3 still hold. The inequality in part (c) of Lemma 3 becomes $\text{Dist}(u) + (j - i)b \leq \text{Dist}(v)$, where the basic lines that contain $p(u)$ and $p(v)$ are along dimensions i and j respectively. Further, in part (d) of Lemma 3 we have $0 \leq \text{Dist}(v) \leq (d-1)b$ instead of $0 \leq \text{Dist}(v) \leq b$. The statement of Lemma 4 remains intact. The inequalities at Lemma 5 now become $t'(u) \leq \max\{t(u)+(\text{Dist}(v)-\text{Dist}(u))+3, t(v')+3\}$ and $t'(u) \leq t(u) + (\text{Dist}(v) - \text{Dist}(u)) + 3$. Lemma 6 holds, using the extended definitions of the quantities involved. Consequently, our schedule for T has makespan $\leq \max\{B, \lceil n/B^d \rceil\} + 24(3d-1)B + 3h + 6d - 1$ and it uses $(12B+1)^d$ processors.

By taking $B = \min\{\lceil n^{1/(d+1)} \rceil, \lceil (n/h)^{1/d} \rceil, \max\{1, \lfloor (m^{1/d} - 1)/12 \rfloor\}\}$, it follows that we can find, in polynomial time, a schedule for T on a d-dimensional mesh with m processors and makespan $O(n/m + n^{1/(d+1)} + h)$. On the other hand, $\Omega(n/m + n^{1/(d+1)} + h)$ is a lower bound on the makespan of any schedule for T on a d-mesh with m processors (for details see Ghosal *et al* [5]). Therefore, our schedule for T is optimal within a constant. ■

These schedules can be generalized to bounded degree tree dags with tasks of arbitrary positive integer execution times and to d–dimensional meshes when the propagation delay of all links is an arbitrary positive integer r_0. In particular, given such a tree dag T and such a mesh, we can find in polynomial time a schedule for the given tree dag on the given mesh with optimal within a constant makespan $O(W/m + (Wr_0^d)^{1/(d+1)} + h_w)$, where W is the sum of the execution times of all the tasks of the tree dag and h_w is its weighted height. These schedules can also be extended to bounded degree forest dags. See [9] for details.

6 Scheduling on Admissible Parallel Architectures

We show how to schedule binary tree dags with unit execution time tasks on any admissible parallel architecture $\mathcal{M}$ with makespan optimal within a constant factor for a large range of the height of those tree dags.

First, we show that a machine of architecture $\mathcal{M}$ can simulate for tree dags the architecture independent model of Papadimitriou and Yannakakis [12] with constant slowdown, for any interprocessor communication delay τ such that $\tau \geq f(\lceil n/\tau \rceil)$. In other words, we find, in polynomial time, a schedule for any binary tree dag T with n unit execution time tasks and height h on a machine of architecture $\mathcal{M}$ with $S(\lceil n/\tau \rceil)$ processors whose makespan is $O(T_{PY})$, where T_{PY} is the makespan of an optimal schedule for T on the architecture independent model of Papadimitriou and Yannakakis [12]. For brevity, we call this architecture independent model of Papadimitriou and Yannakakis [12] the *PY model*.

Second, by choosing appropriate values for τ, and by using two upper bounds on T_{PY} in [8, 13], we provide a polynomial time computable schedule for T on a machine of an admissible parallel architecture $\mathcal{M}$ with an optimal within a constant factor makespan

$T_{\max} = O(g(n) + h)$, when $h \notin (g(n)n^{-\epsilon}, g(n)\log n)$. The number of processors used is optimal within a constant factor when $h \leq g(n)n^{-\epsilon}$, and is optimal within a $O(\log n)$ factor when $h \geq g(n)\log n$. Further, given a positive integer m, we provide a polynomial time computable schedule for T on a machine of an admissible parallel architecture $\mathcal{M}$ with $S(m)$ processors and an optimal within a constant makespan $O(g(n) + n/m)$, when $h \leq g(n)n^{-\epsilon}$. We note that our schedules for admissible architectures generalize to the case of bounded degree forest dags with tasks with arbitrary positive integer execution times, and to admissible architectures when the propagation delay of all the links is a given arbitrary positive integer.

6.1 Simulating the PY Model by Parallel Architectures

Given a parallel architecture $\mathcal{M}$ we show that, for any binary tree dag, $\mathcal{M}$ can simulate with constant slowdown the PY model with a suitable parameter τ. The PY model [12] is defined by having the value of a task, whose execution is completed by a processor p at time t, available to processor p at time t, and to any other processor at time $t + \tau$. Because in a parallel architecture links have constant bandwidth, because an efficient schedule for the PY model may use too many processors, and because the PY model over-estimates the interprocessor communication delay by taking it to always be the machine diameter, the schedule provided by the method in [12] does not necessarily provide an efficient schedule for that architecture. Careful placement of tasks, in order to limit link contention, while making sure that the number of processors used is not too large, is required.

An outline of this section is as follows. Using a function introduced by Papadimitriou and Yannakakis [12], we decompose a tree dag into subtrees. We then number those subtrees in a certain order. Subsequently, using an Eulerian tour of a certain multigraph associated with a parallel machine, we index all the processors. Using those numbered subtrees above and that indexing of the processors, we describe the assignment of tasks to processors, the execution of tasks by processors, and the routing of values of tasks to their destinations. Finally, we compute an upper bound on the time to execute all tasks according to our schedule.

A tree dag T with n unit execution time tasks and height h is given. The e function introduced by Papadimitriou and Yannakakis [12], induces a decomposition of T into subtrees as follows. Compute the e value for each task u of T as in [12]. Let T_1 be the subgraph of T that consists of the root of T together with its $\min\{k, \tau\}$ ancestors in T with the highest e values, where k is the number of ancestors of the root of T. Since the e function is increasing along any directed path in T, it follows that T_1 is a subtree of T. Then, using the same approach recursively, decompose each subtree in the forest $T - T_1$. The resulting subtrees $T_1, T_2, T_3, \ldots$ form a decomposition of T into subtrees. We call this decomposition of T a *PY–decomposition* of T.

To a PY–decomposition of T there corresponds a compressed tree T' as follows. For each subtree T_i there is a supernode u_i in T', and we say that supernode u_i represents subtree T_i. There exists an edge in T' from a supernode u_i to a supernode u_j if the successor of the root of the subtree T_i is in the subtree T_j. The level of a supernode u_i is equal to the distance of u_i from the root of T'. The height of a supernode u_i is equal to the height of the subtree of T' that is rooted at u_i. A supernode of height 0 is a leaf supernode.

We number the supernodes of T' as follows. Fix any left-to-right order for the predecessors of each supernode. Perform a preorder traversal of T', by viewing it as an undirected rooted tree, and number its supernodes with consecutive positive integers according to the order they are visited and so that the root supernode is numbered 1. The preorder number of a supernode u_i is the number assigned to it in this manner. Let $\mathrm{rma}(u_i)$ be the supernode with the maximum preorder number in the subtree of T' that is rooted at supernode u_i. Supernode $\mathrm{rma}(u_i)$ is the rightmost leaf ancestor of supernode u_i in T'.

Let M be a machine of a parallel architecture $\mathcal{M}$ with $m = S(\lceil n/\tau \rceil)$ processors. Index the processors of M as follows. Find a spanning tree M' of M with diameter $\leq 2f(m)$. Construct an Eulerian multigraph M'' by using two copies of each edge of M', and then find an Euler tour of M''. Number the processors of M with consecutive positive integers, starting from 1, according to the order in which each processor was encountered for the first time on that Euler tour. The *index* of a processor is equal to the number assigned to it in this manner. Let $M'[i,j]$ denote the smallest subtree of M' that spans all processors whose index is in the interval $[i,j]$, $1 \leq i \leq j \leq m$.

For simplicity, we describe our schedule for T on machine M with respect to supernodes of the compressed tree T'. We make the following (intuitive) conventions. Let u_i be any supernode of T', T_i be the subtree of T represented by u_i, and p be any processor of M. The value of supernode u_i is the value of the root task of T_i. We say that supernode u_i has been assigned to processor p if all the tasks in T_i have been assigned to p. We say that processor p executes supernode u_i whenever p executes all the tasks in T_i (in any order that respects precedence constraints among tasks), provided that the values of all predecessor supernodes of u_i are available to p.

We now describe our schedule for T on M. Assign the supernodes of T' to processors of M as follows. A processor is *available* if it has been assigned less than $\tau + 1$ tasks. Let $u_1, u_2, u_3, \ldots$ be the supernodes of T' in increasing order of their preorder numbers. For $i = 1, 2, 3, \ldots$, assign supernode u_i to the lowest index available processor of M. Let $p(u_i)$ be the index of the processor that has been assigned u_i. Processors execute the supernodes assigned to them level–by–level starting from supernodes of level h' as in [5], where h' is the height of T'. In particular, for $k = h'$ down to zero perform the following two steps:

Step 1: Each processor executes all supernodes of level k that have been assigned to it.

Step 2: Each processor routes the value(s) of the supernode(s) it executed at

Step 1 to the processor assigned their successor supernode u in T'. Routing is done over shortest paths in $M'[p(u), p(\mathrm{rma}(u))]$ and link contention is resolved in FIFO order.

Because the root supernode has no successors, step 2 is not performed for $k = 0$.

We note here that compressed trees, preorder traversal, level–by–level execution, and routing over non–overlapping intervals were used in Ghosal *et al* [5]. However, they use a different compressed tree. Our approach is a generalization of the approach used in [8] to simulate, for tree dags, the PY model by a linear array.

We state now the main theorem of this section.

Theorem 3 *Let T be a binary tree dag with n unit execution time tasks, and $\mathcal{M}$ be a parallel architecture. Let τ be an integer such that $\tau \geq f(\lceil n/\tau \rceil)$. Then, we can find, in polynomial time, a schedule for T on a machine of architecture $\mathcal{M}$ with $S(\lceil n/\tau \rceil)$ processors whose makespan is $O(T_{PY})$, where T_{PY} is the makespan of an optimal schedule for T on the PY model with parameter τ.*

Proof: First, we prove that at the kth iteration of step 2, $k \geq 1$, the value of each supernode of level k is routed to the processor that has been assigned its successor supernode. Let v be a supernode of level k and let u be its successor supernode. Since supernodes are assigned to processors according to their preorder number and since the preorder number of v is between the preorder numbers of u and $\mathrm{rma}(u)$, it follows that $p(u) \leq p(v) \leq p(\mathrm{rma}(u))$. Hence, v has been assigned to a processor in $M'[p(u), p(\mathrm{rma}(u))]$, and its value is routed to $p(u)$ at the kth iteration of step 2. Further, in Lemma 8 below, we prove that all supernodes of level k that are assigned to the same processor have the same successor supernode. Consequently, the values of all supernodes of level k are available to their successor supernodes for the next iteration of step 1.

Second, in Lemma 7 below, we prove that the height h' of the compressed tree T' is $\leq \lceil e(r)/\tau \rceil$, where r is the root of T. Hence, the total number of iterations of steps 1 and 2 is $\leq \lceil e(r)/\tau \rceil + 1$.

Third, in Lemma 9 below, we prove that each iteration of step 1 takes $\leq 2\tau + 1$ time, and each iteration of step 2 takes $\leq 2f(\lceil n/\tau \rceil) + 4\tau + 4$ time. Since $m = S(\lceil n/\tau \rceil)$, $f(m) \leq \tau$. Thus, each iteration of steps 1 and 2 takes no more than $8\tau + 5$ time.

Since the total number of iterations is $\leq e(r)/\tau + 2$, the time to execute T on M is no more than $13e(r) + 16\tau + 10$. On the other hand, Papadimitriou and Yannakakis [12] show that $e(r) \leq T_{PY}$. Therefore, the makespan of our schedule for T on M is $O(T_{PY})$. ■

In the remainder of this section we prove Lemmas 7, 8, and 9.

Lemma 7 *The height of the compressed tree T' is $\leq \lceil e(r)/\tau \rceil$, where r is the root of T. Each non–leaf supernode of T' represents a subtree of T with $\tau + 1$ tasks, and each leaf supernode represents a subtree of T with $\leq \tau + 1$ tasks. The number of non–leaf supernodes is $\leq \lceil n/(\tau + 1) \rceil$.*

Proof: Consider two supernodes u_i and u_j such that u_i is a predecessor of u_j in T'. Let r_i and r_j be the root tasks of the subtrees T_i and T_j represented by u_i and u_j respectively. Since the e function is increasing along any directed path in T, and since T_j contains r_j together with its τ ancestors with the highest e values, it follows that $e(r_j) \geq e(r_i) + \tau$. In addition, T_j contains $\tau + 1$ tasks, while T_i contains $\leq \tau + 1$ tasks. Consequently, the height of T' is $\leq \lceil e(r)/\tau \rceil$, a non–leaf supernode represents a subtree of T with $\tau + 1$ tasks, and a leaf supernode represents a subtree of T with $\leq \tau + 1$ tasks. Furthermore, since T has n tasks, T' has $\leq \lceil n/(\tau + 1) \rceil$ non–leaf supernodes. ■

Lemma 8 *There are enough processors to assign all supernodes of T'. Each processor is assigned at most one non–leaf supernode. Each processor is assigned at most $2\tau + 1$ tasks. Further, all leaf supernodes assigned to the same processor have the same successor supernode. If a processor is assigned a leaf supernode and a non–leaf supernode then either they both have the same successor supernode or they have different levels.*

Proof: Consider the assignment of supernodes to processors. Since M has $m = S(\lceil n/\tau \rceil) \geq \lceil n/(\tau + 1) \rceil$ processors and a processor is available if it has been assigned $< \tau + 1$ tasks, there are enough processors to assign all the supernodes of T'. In addition, since a non–leaf supernode represents a subtree with $\tau + 1$ tasks, each processor is assigned at most one non–leaf supernode, and no processor is assigned more than $2\tau + 1$ tasks. Observe that if u_j is the successor supernode of a supernode u_i then $p(u_j) \leq p(u_i)$. Moreover, since supernodes are assigned to processors in preorder, if a processor has been assigned more than one supernode then all those supernodes, except possibly one supernode which is a non–leaf supernode, are leaf supernodes and have the same successor supernode. Consider now a processor that has been assigned a leaf supernode u_i and a non–leaf supernode u_j. Note that the preorder number of u_i is less than the preorder number of u_j. Consequently, either u_i and u_j have the same successor supernode or u_i and u_j have different levels. ■

Lemma 9 *The time to perform step 1 is no more than $2\tau + 1$ and the time to perform step 2 is no more than $2f(\lceil n/\tau \rceil) + 4\tau + 4$.*

Proof: By Lemma 8, each processor is assigned $\leq 2\tau + 1$ tasks. Therefore, each iteration of step 1 takes time $\leq 2\tau + 1$.

Consider now the kth iteration of step 2, $k = h', h' - 1, \ldots, 2, 1$. At that iteration, values of supernodes of level k are communicated to supernodes of level $k - 1$. Let $v_1, v_2, v_3, \ldots$

be the non–leaf supernodes of T', in increasing order of their preorder numbers, whose level is $k-1$. Consider one such supernode v_i. All predecessor supernodes of v_i are at level k and have preorder numbers between the preorder numbers of the supernodes v_i and $\mathrm{rma}(v_i)$. Because supernodes are assigned to processors according to their preorder numbers, all predecessor supernodes of v_i have been assigned to processors with indices in the interval $[p(v_i), p(\mathrm{rma}(v_i))]$. The value of each predecessor supernode v_i' of v_i is routed to the processor assigned v_i using a shortest path in $M'[p(v_i), p(\mathrm{rma}(v_i))]$. Since link contention is resolved in FIFO order, the communication delay to route the value of v_i' to v_i is the sum of the delay due to distance traveled and the delay due to link contention. Because the diameter of M' is $\leq 2f(m) = 2f(\lceil n/\tau \rceil)$, the time to route the value of v_i' to v_i, that is due only to distance traveled, is $\leq 2f(\lceil n/\tau \rceil)$. (Recall that, by definition of S and f, $f(m) = f(S(m))$.)

Next, we find the delay for routing the value of v_i' to v_i that is due only to link contention. The value of v_i' may compete for using a link with values of supernodes destined to v_i and to other supernodes of level $k-1$. Since v_i represents a subtree of a binary tree dag with $\tau+1$ tasks, the number of predecessor supernodes of v_i is $\leq 2(\tau+1)$. Hence, the value of v_i' may compete with $\leq 2(\tau+1)$ other values that are destined to v_i. In addition, the number of values that are routed in $M'[p(v_i), p(\mathrm{rma}(v_i))]$ is at most $2(\tau+1)$. It remains to analyze the effect of link contention due to values destined to other supernodes of level $k-1$. For any two such supernodes v_j, v_l, $j < l$, we have that the preorder number of $\mathrm{rma}(v_j)$ is less than that of v_l. Consequently, $p(\mathrm{rma}(v_j)) \leq p(v_l)$, which implies that all the intervals $[p(v_1), p(\mathrm{rma}(v_1))]$, $[p(v_2), p(\mathrm{rma}(v_2))]$, $[p(v_3), p(\mathrm{rma}(v_3))]$, ... are non–overlapping. Further, since the processors are indexed with respect to an Euler tour of a double–copy of M', it follows that each link of M' is contained in at most two subgraphs $M'[p(v_j), p(\mathrm{rma}(v_j))]$ of M', for $j = 1, 2, 3, \ldots$. Therefore, the value of v_i' competes with at most $4(\tau+1)$ other values. Consequently, the communication delay to route the value of v_i' to v_i is $\leq 2f(\lceil n/\tau \rceil) + 4(\tau+1)$, and the lemma follows. ■

We mention here that our simulation method can be generalized, in a straightforward manner, to bounded degree forest dags with tasks of arbitrary positive integer execution times and to architectures when the propagation delay of all the links is an arbitrary positive integer (see [9] for details).

6.2 Optimal within a Constant Schedules for Tree Dags on Parallel Architectures

Using the simulation result of Theorem 3, two upper bounds in [8, 13] on the makespan of a schedule for a binary tree dag on the PY model, a lower bound on the makespan of a schedule for a tree dag on an admissible architecture, and by choosing appropriate τ values, we provide schedules for binary tree dags on any admissible parallel architecture.

We state now two upper bounds in [8, 13] on the makespan of a binary tree dag on the

PY model. Thurimella and Yesha [13] prove that there exists a schedule for a binary tree dag with n unit execution time tasks and height h on the PY model with parameter τ whose makespan is $\leq h + \tau\lceil \log n \rceil$. In [8], we prove the following lemma that provides an upper bound on the makespan of schedules for tree dags on the PY model.

Lemma 10 *Let T be a binary tree dag with n tasks and height h. Let τ be a positive integer such that $n > \tau > h$. Then, there exists a schedule for T on the PY model with parameter τ whose makespan is $O(\tau \log(n/\tau)/\log(\tau/h))$.*

Proof sketch: Using the edge–centroid decomposition method recursively we construct a sequence of binary tree dags D_i from T (we take $D_0 = T$) such that the number of tasks in D_i, as i increases, decreases geometrically by a factor of $\Omega(h/\tau)$. Then, for each i, we show that we can execute all the tasks in $D_i - D_{i-1}$ on the PY model in $O(\tau)$ time. For details see [8]. ■

We continue by proving the following lower bound on the makespan of a schedule for a tree dag on an admissible parallel architecture.

Lemma 11 *Let T be a tree dag with n unit execution time tasks and height h. Let $T_{\max}$ be the makespan of a schedule for T on an admissible parallel architecture $\mathcal{M}$. Then, $T_{\max} = \Omega(g(n) + h)$ and the number of processors used is $\Omega(n/T_{\max})$.*

Proof: Consider an optimal schedule for T on a machine M of architecture $\mathcal{M}$. That schedule uses a subgraph $\widehat{M}$ of M with $k \geq 1$ processors. Since $\mathcal{M}$ is admissible, the diameter of $\widehat{M}$ is $\Omega(f(k))$. The makespan $T_{\max}$ of that schedule is $\geq \lceil n/k \rceil$, because at least one processor has been assigned $\geq \lceil n/k \rceil$ tasks. Since both the height of T and half the diameter of $\widehat{M}$ is a lower bound on the makespan, it follows that the makespan of that schedule is $\Omega(n/k + f(k) + h)$. On the other hand, by definition of g and since f is non–decreasing, we have that if $\lceil n/k \rceil \leq g(n)$ then $f(k) \geq g(n)$. Thus, $\max\{\lceil n/k \rceil, f(k)\} \geq g(n)$ for all $k \geq 1$. Consequently, the makespan for that schedule is $\Omega(g(n) + h)$. Further, any schedule for T on $\mathcal{M}$ with makespan $T_{\max}$ uses $\Omega(n/T_{\max})$ processors. ■

Using the upper and lower bounds above, Theorem 3, and choosing appropriate τ's, we prove the following two theorems.

Theorem 4 *Let ϵ be any fixed positive real number. Then, for any binary tree dag T with n unit execution time tasks and height $h \notin (g(n)n^{-\epsilon}, g(n)\log n)$, we can find, in polynomial time, a schedule for T on $\mathcal{M}$ with $O(\min\{n/g(n), n\log n/h\})$ processors and optimal within a constant factor makespan $O(g(n)+h)$. This schedule is processors–optimal within a $O(\log n)$ factor when $h \geq g(n)\log n$, and is processors–optimal within a constant factor when $h \leq g(n)n^{-\epsilon}$.*

Proof: There are two cases to consider.

Case 1: $h \geq g(n)\log n$. Take $\tau = \lceil h/\log n \rceil$. Since $T_{PY} \leq h + \tau\lceil \log n \rceil$ [13], $T_{PY} = O(h)$. Further, since $\tau \geq g(n)$ and f is non-decreasing, $f(\lceil n/g(n) \rceil) \geq f(\lceil n/\tau \rceil)$. By definition of g, $g(n) \geq f(\lceil n/g(n) \rceil)$, implying that $\tau \geq f(\lceil n/\tau \rceil)$. Hence, by Theorem 3, we can find, in polynomial time, a schedule for T on $\mathcal{M}$ with $S(\lceil n/\tau \rceil) = O(n \log n/h)$ processors and makespan $O(h)$.

Case 2: $h \leq g(n)n^{-\epsilon}$. Take $\tau = g(n)$. By Lemma 10, $T_{PY} = O(\tau \log(n/\tau)/\log(\tau/h)) = O(\tau) = O(g(n))$. By definition of g and since f is non-decreasing, $\tau \geq f(\lceil n/\tau \rceil) \geq f(\lceil n/(\tau+1) \rceil)$. Thus, by Theorem 3, we can find, in polynomial time, a schedule for T on $\mathcal{M}$ with $S(\lceil n/\tau \rceil) = O(n/g(n))$ processors and makespan $O(g(n))$.

The theorem now follows from Lemma 11. ■

Theorem 5 *Let ϵ be any fixed positive real number. Then, for any binary tree dag T with n unit execution time tasks and height $h \leq g(n)n^{-\epsilon}$, we can find, in polynomial time, given any positive integer m, a schedule for T on $\mathcal{M}$ with* $\min\{S(\lceil n/g(n) \rceil), S(m)\}$ *processors and makespan $O(g(n)+n/m)$, i.e. both optimal and processors–optimal within a constant factor.*

Proof: There are two cases to consider.

Case 1: $m < n/g(n)$. Take $\tau = \lceil n/m \rceil - 1$. Since $\lceil n/\tau \rceil \leq m < \lceil n/g(n) \rceil$ and f is non-decreasing, $f(\lceil n/\tau \rceil) \leq f(m) \leq f(\lceil n/g(n) \rceil$. Since $g(n) < n/m$, $g(n) \leq \lceil n/m \rceil - 1 = \tau$. By definition of g, $g(n) \geq f(\lceil n/g(n) \rceil)$, implying that $\tau \geq f(\lceil n/\tau \rceil)$. Besides, since $g(n) \leq \tau$, $h \leq \tau n^{-\epsilon}$. By Lemma 10, $T_{PY} = O(\tau) = O(n/m)$. Hence, by Theorem 3, we can find, in polynomial time, a schedule for T on $\mathcal{M}$ with $S(\lceil n/\tau \rceil) \leq S(m) \leq S(\lceil n/g(n) \rceil)$ processors and makespan $O(n/m)$.

Case 2: $m \geq \lceil n/g(n) \rceil$. Take $\tau = g(n)$. By definition of g and since $\tau = g(n)$, $\tau \geq f(\lceil n/g(n) \rceil)$. By Lemma 10 and since $h \leq g(n)n^{-\epsilon}$, $T_{PY} = O(g(n))$. Hence, by Theorem 3, we can find, in polynomial time, a schedule for T on $\mathcal{M}$ with $S(\lceil n/g(n) \rceil) \leq S(m)$ processors and makespan $O(g(n))$.

The theorem now follows from Lemma 11. ■

As an application of Theorems 4 and 5, we provide polynomial time computable schedules for binary tree dags on hypercube and complete binary tree architectures with optimal within a constant factor makespan and time–processors product optimal within a $O(\log n)$ factor. It can be shown easily that for hypercubes and complete binary tree architectures we have $S(m) = \Theta(m)$, $f(m) = \Theta(\log m)$ and $g(n) = \Theta(\log n)$. Moreover, both architectures are admissible. Then, using Theorem 4, we can find, in polynomial time, a schedule for a tree dag T with n tasks and height $h = \Omega(\log^2 n)$ on a hypercube or complete binary tree architecture with $O(n(\log n)/h)$ processors and optimal within a constant factor makespan $O(h)$. Note that, for hypercubes, one can obtain, via an embedding result in Bhatt *et al* [2], a

schedule for a binary tree dag on a hypercube with makespan $O(h)$, but with much more processors, namely n.

Further, it can be shown, by a straightforward extension of a result in [8], that for arbitrarily many positive integers n there exist binary tree dags with n unit execution time tasks and height $o(n^{1/(d+1)})$ that require $\omega(n^{1/(d+1)})$ time on the PY model. Therefore, our optimal within constant schedules for tree dags on d-dimensional meshes can not be derived by simulating schedules for the PY model with the interpretation for the interprocessor communication delay τ implied by Papadimitriou and Yannakakis [12].

Theorems 3, 4, and 5 can be generalized to bounded degree forest dags with tasks of arbitrary positive integer execution times and to admissible architectures when the propagation delay for all the links is an arbitrary positive integer (see [9] for details).

References

[1] A. Aggarwal, A. K. Chandra, and M. Snir. Communication complexity of PRAMs. *Theoretical Computer Science*, 71, pp. 3–28, 1990.

[2] S. Bhatt, F. Chung, T. Leighton, and A. Rosenberg. Optimal Simulations of Tree Machines. In *Proc. of 27th Annual Symposium on Foundations of Computer Science*, pp. 274–282, 1986.

[3] G. Frederickson. Updating of minimum spanning trees, with applications. *SIAM Journal on Computing*, 14(4), pp. 781–798, 1985.

[4] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, San Francisco, 1979.

[5] D. Ghosal, A. Mukherjee, R. Thurimella, and Y. Yesha. Mapping task trees onto a linear array. In *Proc. of 1991 International Conference on Parallel Processing*, Vol. I, pp. 629–633, 1991.

[6] D. Ghosal, A. Mukherjee, R. Thurimella, and Y. Yesha. Scheduling task-trees onto a linear array. manuscript, 1992.

[7] H. Jung, L. Kirousis, and P. Spirakis. Lower bounds and efficient algorithms for multiprocessor scheduling of dags with communication delays. In *Proc. of ACM Symposium on Parallel Algorithms and Architectures*, pp. 254–264, 1989.

[8] K. Kalpakis and Y. Yesha. On the Power of the Linear Array Architecture for Performing Tree-Structured Computations. UMIACS-TR-92-75 CS-TR-2924, University of Maryland at College Park, Institute for Advanced Computer Studies, July 1992.

[9] K. Kalpakis and Y. Yesha. Optimal within a Constant Schedules for Forest Dags on Parallel Architectures. UMIACS-TR-92-110.1 CS-TR-2974.1, University of Maryland at College Park, Institute for Advanced Computer Studies, October 1992.

[10] S. R. Kosaraju. Parallel evaluation of division-free arithmetic expressions. In *Proc. of 18th Annual ACM Symposium on Theory of Computing*, pp. 231–239, 1986.

[11] C. H. Papadimitriou and J. D. Ullman. A communication-time tradeoff. *SIAM Journal on Computing*, 16(4), pp. 639–646, 1987.

[12] C. H. Papadimitriou and M. Yannakakis. Towards an architecture-independent analysis of parallel algorithms. *SIAM Journal on Computing*, 19(2), pp. 322–328, 1990.

[13] R. Thurimella and Y. Yesha. A scheduling principle for precedence graphs with communication delay. In *Proc. of 1992 International Conference on Parallel Processing*, Vol. III, pp. 229–236, 1992.

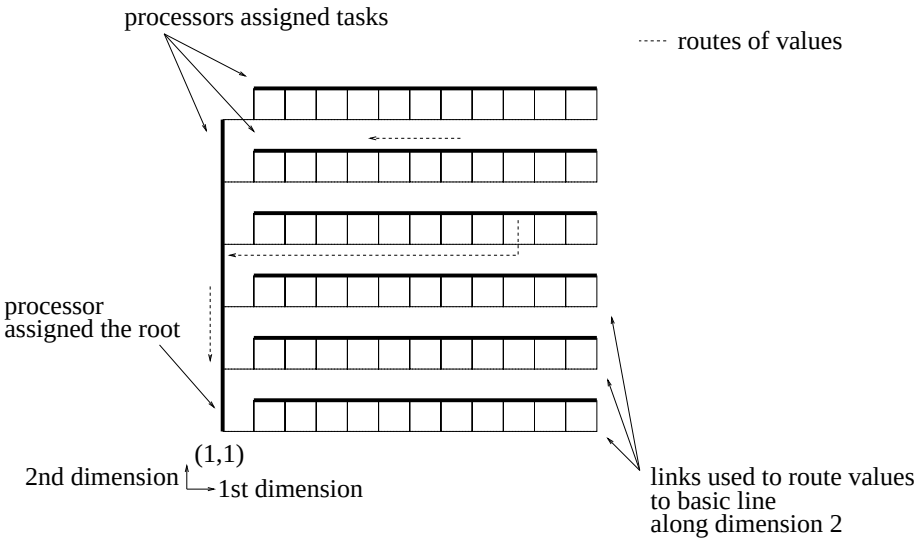

Figure 1: Scheduling a tree on a 2-dimensional mesh. Tasks are assigned to processors on basic lines (thick lines). Links on thin lines are used only for routing values of tasks from processors on basic lines along dimension 1 to processors on the basic line along dimension 2. The root of that tree is assigned to processor $(1, 2)$. The possible routes of the value of a task are shown with dashed lines.

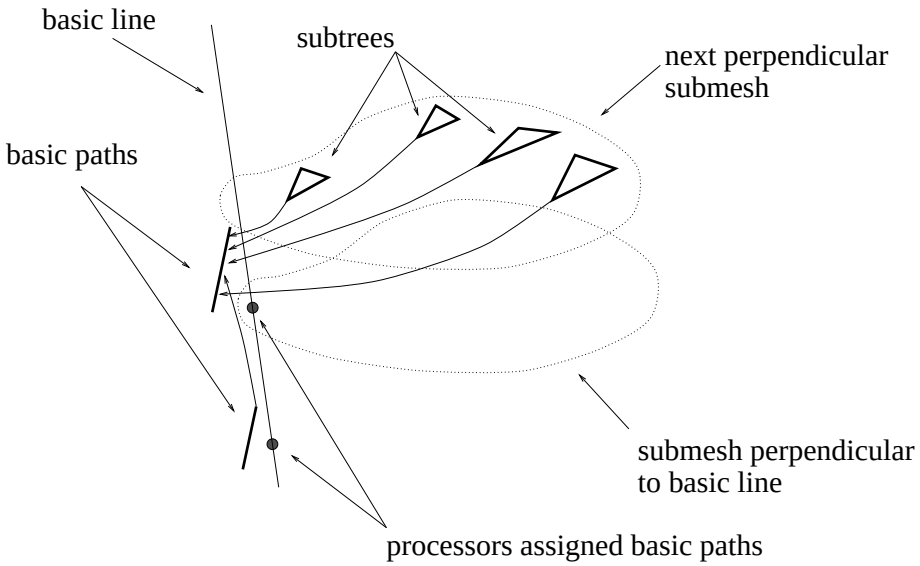

Figure 2: The task assignment method. Basic paths π are assigned on a basic line along some dimension. For each basic path π some subtrees of tasks whose successors are on π are assigned to a submesh that is perpendicular to that basic line.

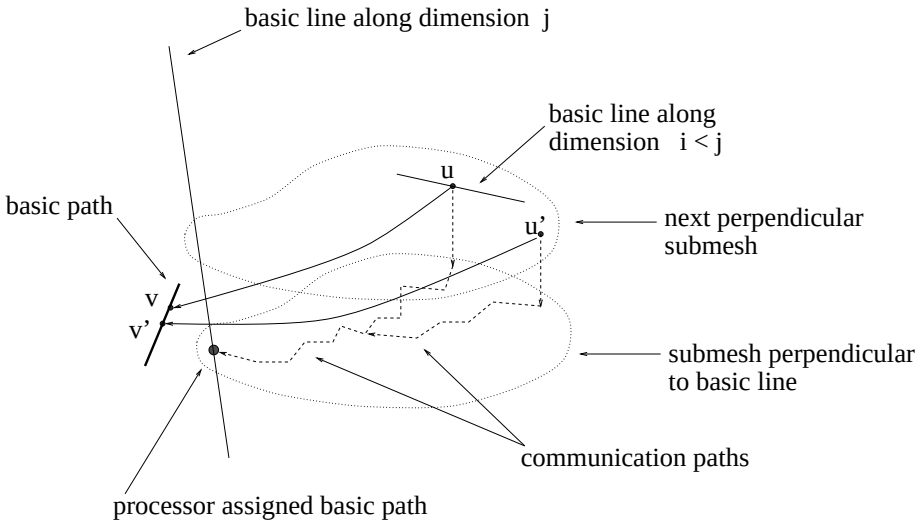

Figure 3: Communicating the value of task u to the processor assigned its successor task v. Tasks u and v have been assigned to different processors on different basic lines.