On the Power of the Linear Array Architecture for Performing Tree-Structured Computations

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

[1]Computer Science Department, University of Maryland Baltimore County, 5401 Wilkens Avenue, Baltimore, MD 21228-5398.

[2]Also, University of Maryland at College Park, Institute for Advanced Computer Studies.

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

Please send all correspondence to:

Dr. Yaacov Yesha
Computer Science Department
University of Maryland Baltimore County
5401 Wilkens Avenue
Baltimore, MD 21228-5398.

Phone: (410) 455-2669 Department Office: (410) 455-3000
Fax: (410) 455-3969
E-mail: yayesha@cs.umbc.edu

Abstract

We consider the problem of scheduling the execution of programs on the linear array architecture. A program is represented by a directed acyclic graph (dag), whose nodes represent tasks, and whose edges represent both precedence constraints and functional dependencies among tasks.

We prove that, for binary tree dags with unit execution time tasks, the linear array can simulate with constant slowdown the architecture independent model of Papadimitriou and Yannakakis [10] when the message-to-instruction ratio is assumed to be equal to the diameter of the linear array. (This is the interpretation implied in [10].) Furthermore, we prove that for binary tree dags with unit execution time tasks the linear array is strictly more powerful than the above interpretation of the model of [10].

Using our simulation result, we give polynomial time algorithms to find schedules for binary tree dags with n tasks and height h on linear arrays, when $h \geq n^{1/2} \log n$ (achieving makespan $O(h)$), and when $h \leq n^{1/2-\epsilon}$ for any fixed $\epsilon > 0$ (achieving makespan $O(n^{1/2})$, where the constant inside the Big Oh depends on ϵ). The makespan achieved by our schedules is within a constant factor of optimal. The time–processors product achieved is within a constant factor of optimal when $h \leq n^{1/2-\epsilon}$, and within an $O(\log n)$ factor of optimal when $h \geq n^{1/2} \log n$. These schedules improve upon the makespan of the schedules in Ghosal *et al* [3] by an $O(\log n)$ factor for those ranges of h, and at the same time also improve upon the time–processors product in [3] by an $O(\log n)$ factor (for $h \leq n^{1/2-\epsilon}$), or maintain the same time–processors product as in [3] up to a constant (for $h \geq n^{1/2} \log n$). Further, for $h \leq n^{1/2}/\log^2 n$, we provide a polynomial time computable schedule on a linear array, that achieves makespan $O(n^{1/2})$ using $O(n^{1/2})$ processors (*i.e.* both optimal up to a constant), this time under an assumption that links have unlimited bandwidth. Using straightforward arguments (see Ghosal *et al* [4]), we extend some of our polynomial time scheduling algorithms to deal with the more general case, where the trees have bounded degree d, and the architecture is a k-dimensional mesh.

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

1 Introduction

An important consideration in mapping the computational structure of an algorithm 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. An algorithm is represented by a directed acyclic graph (dag), whose nodes represent tasks, and whose edges represent both precedence constraints and functional dependencies among tasks. Given the precedence graph of an algorithm and the architecture of a multiprocessor system, the problem of finding an efficient mapping which minimizes the computation and communication time is, in general, a difficult computational problem and has been the target of extensive research in recent years.

The problem considered in this paper is the problem of finding efficient mappings of tree structured algorithms onto linear arrays of processors. Trees are the computational structure of many important classes of algorithms such as divide-and-conquer. The linear array of processors architecture is an important regular bounded-degree architecture suitable for VLSI and Wafer Scale Integration multiprocessor systems [7, 8]. Further, its simplicity has some practical implementation-related advantages compared to more complex architectures. However, the communication overhead in a linear array grows linearly with the number of processors it has. This makes the task of designing efficient algorithms for linear arrays a challenging and a non-trivial one.

Papadimitriou and Ullman [9] prove non-trivial lower bounds on the communication c, the communication delay d, and the computation time t required to evaluate an $n \times n$ diamond dag. They show that the following two conditions must hold: $(c+n)t = \Omega(n^3)$ and $(d+1)t = \Omega(n^2)$. Depending on the parallel machine architecture one of these conditions is more significant.

Papadimitriou and Yannakakis [10] study the problem of evaluating the performance of an algorithm, rendered as a dag, on an abstract parallel machine. Their model, that we call *the PY model*, consists of a dag, an unlimited number of processors, and a parameter τ capturing the interprocessor communication delay in a parallel machine. It is implied in [10] that τ is equal to the diameter of an actual parallel machine. Their objective is, given a dag and parameter τ, to find a schedule for that dag which takes the minimum time to execute it. They prove that the problem of finding such a schedule is NP-complete and they provide an efficient approximation algorithm for this problem with performance ratio of 2. This approximation algorithm, which we call the *PY method*, is based on a simple function ϵ evaluated at all nodes of the input dag.

The linear array model is different from the PY model, because there is no notion of limited bandwidth in the PY model, and because in the linear array the interprocessor communication delay is not the same for all pairs of communicating processors.

Jung, Kirousis, and Spirakis [5] give an exact dynamic programming algorithm, running

in $O(n^{\tau+1})$ time, to find an optimal schedule for a dag with n tasks on the PY model with parameter τ.

Ghosal, Mukherjee, Thurimella, and Yesha [3] consider the problem of scheduling a tree dag T with n unit-time tasks and height h on a linear array. They give a polynomial time algorithm to find a schedule for such a dag on a linear array with $O(n/\max\{\sqrt{n}, h\})$ processors. The makespan of their schedules is $O(\max\{\sqrt{n}, h\} \log n)$. [4]

Aggarwal, Chandra, and Snir [1] consider the Local-memory PRAM model (LPRAM) to capture communication and computation requirements of algorithms on parallel machines. The LPRAM model is a CREW PRAM [5] with each processor having its own local memory, where the shared memory has latency l. They prove that for any binary tree dag T, where each node has 0 or 2 children, with n nodes and height h, the number of computation steps required to compute T by a LPRAM with p processors is $\Theta(n/p + h)$, while the number of communication steps is $\Omega((n/p) + \log n + \sqrt{h}) \cap O((n/p) + \min\{\sqrt{n}, h\})$. To apply this model to the linear array, one will take $l = \Theta(p)$. Then, the total time to schedule that tree will be $\Theta(n/p + h) + l\Omega((n/p) + \log n + \sqrt{h}) \cap O((n/p) + \min\{\sqrt{n}, h\}) = \Omega(n + p \log n + p\sqrt{h}) \cap O(n + p \min\{\sqrt{n}, h\})$. That is, the time is $\Omega(n)$. The major difference between the LPRAM model and the models in this paper is that the LPRAM model does not allow pipelining. Further, similar to the PY model, all communication steps take the same time l.

We prove that, for binary tree dags with unit execution time tasks, the linear array can simulate with constant slowdown the architecture independent model of Papadimitriou and Yannakakis [10] when the message-to-instruction ratio is assumed to be equal to the diameter of the linear array. This is the interpretation implied in [10]. This interpretation induces a model that we call the LIN–PY model. Furthermore, we prove that for binary tree dags with unit execution time tasks the linear array is strictly more powerful than the above interpretation of the model of [10] even though the linear array has unit bandwidth links.

We apply the above simulation result to obtain polynomial time computable schedules for trees on linear arrays, that achieve makespan within a constant factor of optimal, for the following important ranges of the tree height h: $h \geq \sqrt{n} \log n$ and $h \leq n^{1/2-\epsilon}$ for any fixed $\epsilon > 0$. The makespan achieved by the schedule for $h \geq \sqrt{n} \log n$ is $O(h)$, which is optimal within a constant, and the number of processors used is $O(n \log n/h)$, which is optimal within a factor of $\log n$. The schedule for $h \leq n^{1/2-\epsilon}$ has makespan $O(\sqrt{n})$, where the constant inside the Big Oh is proportional to $1/\epsilon$, and uses $O(\sqrt{n})$ processors, both being optimal within a constant factor when ϵ is fixed. These two results improve upon the makespan of the schedules in Ghosal *et al* [3] for those ranges of h, and at the same time also improve upon the time–processors product in [3] (for $h \leq n^{1/2-\epsilon}$), or maintain the same time–processors product as in [3] up to a constant (for $h \geq \sqrt{n} \log n$). We also prove that, for $h < \sqrt{n}$, our schedules achieve makespan $O(\sqrt{n} \log n / \log(\sqrt{n}/h))$ and use $O(\sqrt{n})$ processors.

[4] Throughout this paper, log denotes the base 2 logarithm.

[5] PRAM stands for Parallel Random Access Machine. CREW means Concurrent Read Exclusive Write.

For the range $h \leq \sqrt{n}/\log^2 n$, we prove an additional result that sheds some light on the makespan complexity of trees on linear arrays. For this range of h, we provide a polynomial time computable schedule on a linear array, that achieves makespan $O(\sqrt{n})$ using $O(\sqrt{n})$ processors (*i.e.* both optimal up to a constant), this time under an assumption that links have *unlimited bandwidth*.

Using straightforward arguments (see Ghosal *et al* [4]), we extend some of our polynomial time scheduling algorithms to deal with the more general case, where the trees have bounded degree d and the architecture is a k-dimensional mesh.

The rest of the paper is organized as follows. In Section 2 we include the needed definitions. In Section 3 we prove that the linear array can simulate, for binary tree dags, the LIN–PY model with constant slowdown. Then, in Section 4, we separate the linear array and the LIN–PY model for binary tree dags by showing that for binary tree dags the linear array is strictly more powerful than the LIN–PY model. In Section 5, we give polynomial time computable, optimal within a constant, schedules for binary tree dags on the linear array for $h \leq n^{1/2-\epsilon}$ and $h \geq \sqrt{n}\log n$, and a polynomial time computable, optimal within a constant, schedule when $h \leq \sqrt{n}/\log^2 n$, this time under an assumption of links of unlimited bandwidth. In Section 6 we give conclusions, and extend some of our results to tree dags of bounded degree, and k-dimensional meshes.

2 Preliminaries

A rooted directed binary tree (tree dag) of n nodes and height h is given. Nodes represent computational tasks, and edges, which are directed towards the root, represent both precedence constraints and functional dependencies among tasks. Given a tree dag T and two tasks u, v in T we say that v is a *child of* u if there exists an edge from v to u in T, and we say that v is an ancestor of u if there exists a (directed) path from v to u in T. Throughout this paper, unless we state otherwise, we assume that each task requires one time unit for execution, and that the task tree is a binary tree.

A linear array is modeled by an undirected graph which is a chain. 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 limited bandwidth. Throughout this paper, unless we state otherwise, we assume that each link has one time unit delay and unit bandwidth.

Tasks are assigned to processors for execution. A task may be assigned to more than one processor, in which case we say that 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*. For simplicity, we will be referring to a copy of a task simply as a task. For a given assignment of tasks, the processors perform computation according to the following 8 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 one time unit 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 time of a schedule, called *makespan* and denoted $T_{\max}$, equals the number of time units that pass until all copies of each task are executed. Given a dag and a machine, a schedule is called *optimal* if its makespan $T_{\max}$ is minimum among all possible schedules for that dag on that machine. Given a dag, a machine, and a time t, a schedule is called *processors–optimal* with respect to t if the number of processors used is minimum among all schedules for that dag on that machine whose makespan is t.

Given a tree dag T with n tasks and height h, our objective is to find a schedule for T on a linear array 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 $T_{\max}$.

It is easy to see that any optimal schedule for T on a linear array has makespan $\Omega(\max\{\sqrt{n}, h\})$ and requires no recomputation [3]. Further, any processors–optimal schedule for T on a linear array with respect to t requires no less than $\lceil n/t \rceil$ processors.

We now describe the architecture independent model introduced by Papadimitriou and Yannakakis [10]. A dag $\mathcal{D} = (V, A)$, whose nodes represent unit execution-time tasks, and whose edges represent both precedence constraints and functional dependencies, is given together with a positive integer τ. A schedule S of $\mathcal{D}$ is a finite set of 3-tuples $S \subset V \times N \times N$, where N denotes the set of non-negative integers, (each 3-tuple $(v, p, t) \in S$ means that processor p executes task v at time t) so that the following three conditions hold:

1. For each $v \in V$ there exist at least one 3-tuple $(v,p,t) \in S$. (Intuitively, each task is executed by some processor.)

2. There are no two 3-tuples (u,p,t),$(u',p,t) \in S$ with $u \neq u'$. (Intuitively, no processor executes more than one task at the same time.)

3. If $(u,v) \in A$ and $(v,p,t) \in S$, then either there is another 3-tuple $(u,p,t') \in S$ with $t' \leq t-1$, or there is another 3-tuple $(u,p',t') \in S$ with $t' \leq t-1-\tau$. (Intuitively, if task v depends on task u, then either u is executed by the same processor p as v at least one time unit earlier, or u is executed by some other processor earlier enough so that the value of u is available to p when p starts executing v.)

The makespan of S equals the largest time appearing in S. We call this model the *PY model* with parameter τ. Papadimitriou and Yannakakis [10] consider the problem of finding, given a dag and a τ, an optimal schedule for that dag on the PY model with parameter τ. They prove that the problem of finding an optimal schedule for a dag on the PY model is a NP-hard problem, and they provide a polynomial time approximation algorithm for this problem whose performance ratio is 2. We call this algorithm the *PY method*. Their algorithm is based on the function $e : V \rightarrow N$ they introduce. For each task u in a dag $\mathcal{D}$, they define $e(u)$ as given below. Let $u_1, u_2, \ldots, u_p$ be all the ancestors of u in $\mathcal{D}$, $e(u_1) \geq e(u_2) \geq \ldots \geq e(u_p)$, for some p. Then, $e(u)$ is given by

$$e(u) = \begin{cases} 0, & \text{if } u \text{ is a source} \\ e(u_k) + k, & \text{otherwise} \end{cases}$$

where $k = \min\{\tau + 1, p\}$. Using this function e, they give a schedule for $\mathcal{D}$ on the PY model with parameter τ, as described below, so that each task u is executed by time 2 $e(u)$. For each task $u \in \mathcal{D}$, u together with its τ ancestors with the highest e value are assigned to one processor. This processor executes no other tasks, and it gets all the values it needs by communicating with other processors. Further, they prove that in any schedule for $\mathcal{D}$ on the PY model with parameter τ, no node $u \in \mathcal{D}$ can be scheduled before time $e(u)$.

Let $f : N \rightarrow N$ be some function. Papadimitriou and Yannakakis [10] imply that, in their model, τ should be taken to be proportional to the diameter of the interconnection network of the actual parallel machine. Motivated by this, we define the *f-PY model* as being the PY model with $\tau = f(m)$, where m is the number of processors used. When f is such that $f(m) = m - 1$, the diameter of a linear array with m processors, for any positive integer m, we call the f-PY model the LIN–PY model. Function f represents the architecture in the following sense: it gives the dependency of the diameter on the number of processors. The LIN–PY model is the interpretation of the PY model for linear arrays.

In this paper we use the edge–centroid tree decomposition method. It is well known that a binary tree with n nodes has an edge whose removal partitions that tree into two subtrees with no more than $\lceil 2n/3 \rceil$ nodes each. This partition is called *edge–centroid decomposition.*

The edge–centroid decomposition can be used recursively to partition a binary tree with n nodes into $O(\sqrt{n})$ subtrees, each with size $O(\sqrt{n})$ [2, 6]. To produce the desired partition we can do the following. Remove an appropriate edge to produce two subtrees, each of size no greater than $\lceil 2n/3 \rceil$. Repeating this procedure on each resulting subtree eventually we will achieve the desired partition. Clearly this partition can be found in polynomial time. It is obvious that the same technique can be used to partition the tree into $O(n/u)$ subtrees, each of size $O(u)$, for any integer u between 1 and n. It is also obvious that the same kind of partition can be obtained for trees of degree bounded by d, except that the constant in the Big Oh notation is proportional to d in this case. Another well known polynomial time computable decomposition, called *node centroid decomposition*, is given by a node of a tree (this time not necessarily binary), whose removal partitions the n-node tree into subtrees, each having no more than $\lceil n/2 \rceil$ nodes.

3 Simulating the LIN–PY Model for Tree Dags by a Linear Array

We describe a slight modified version of the PY method for tree dags. Then, using that modified PY method, we show how to construct a schedule for a tree dag T with n tasks and root r on a linear array with m processors, $\lceil n/(2\tau+1) \rceil \leq m \leq \lceil n/(\tau+1) \rceil$, and with makespan no more than $4e(r) + \lceil n/(\tau+1) \rceil$. That is, the linear array can simulate the LIN–PY model for tree dags with constant slowdown. We note that, because there is no notion of limited bandwidth in the PY model, schedules for tree dags on the PY model with parameter τ can not be directly used as schedules of these tree dags on a linear array.

We start by describing our modification to the PY method for scheduling tree dags on the PY model with parameter τ. Consider the PY method for the PY model with parameter τ. We show how tasks can be assigned to processors of the PY model with parameter τ, so that there is no recomputation and the makespan is still within a factor of 2 of optimal. Let T' be the subtree of T which contains r and the τ ancestor tasks of r with the highest e values. Assign T' to one processor. This processor will execute only the tasks in T' and it will receive all the other values it needs by communicating with other processors. Then, recursively, using the same approach, assign each subtree in the forest $T - T'$ to processors. Each processor executes its tasks according to their e values. When a processor finishes executing its subtree of tasks, it communicates the value of the root of its subtree to the processor which has been assigned the successor of its root. The time needed to execute T, using the assignment of tasks to processors above, is no more that $2e(r)$ as we prove in the next lemma. We call this method, for constructing a schedule for T on the PY model with parameter τ, the *modified PY method*. It is easy to see that the modified PY method runs in polynomial time.

Lemma 1 *Let T be a tree dag (not necessarily binary) with root r. Let S be the schedule for T on the PY model with parameter τ, which is produced by the modified PY method. Then, S has no recomputation, and each processor in S is assigned to execute a subtree of T. Further, the makespan of S is no more than $2e(r)$.*

Proof: We prove this lemma by induction on the number of tasks n of T.

Basis. The lemma is trivially true for all trees T with $\leq \tau + 1$ tasks, because for these trees T the schedule S uses only one processor which executes all the tasks.

Inductive hypothesis. Suppose the lemma is true for all trees T with $< n$ tasks, $n > \tau + 1$.

Inductive step. We prove the lemma for any tree T with n tasks. Let u_0 denote the root of T. Let $u_1, u_2, \ldots, u_{n-1}$ be all the ancestors of u_0 so that $e(u_0) \geq e(u_1) \geq e(u_2) \geq \ldots \geq e(u_{n-1})$. Note that, by definition of e, $e(u_0) = e(u_{\tau+1}) + \tau + 1$. Further, function e is strictly monotonically increasing along any directed path of T.

In the modified PY method all the tasks in $\{u_0, u_1, \ldots, u_\tau\}$ are executed by one processor and receive the values they need by communicating with other processors. Let T' be the subgraph of T that is induced by the tasks in $\{u_0, u_1, \ldots, u_\tau\}$. Each tree in the forest $T - T'$ is recursively scheduled using the modified PY method.

We claim that the subgraph T' of T is a subtree of T. We prove this claim by contradiction. Suppose that T' is not a subtree of T. Then, there must be a task $u_i \in T'$ such that the immediate successor u_i' of u_i is not in T'. Since $e(u_i') > e(u_i)$, from the definition of T' it follows that u_i' must be in T'. This is a contradiction. Thus, T' is subtree of T.

Because each tree T_i in the forest $T - T'$ has less than n tasks, from the inductive hypothesis it follows that the schedule for T_i, produced by the modified PY method, has no recomputation, and that each processor is assigned a subtree of T_i. Further, since each T_i is a subtree of T, the schedule for $T - T'$ has no recomputation, and each processor is assigned a subtree of T.

Let $r_1, r_2, \ldots, r_k$, be the roots of the subtrees in the forest $T - T'$. From the inductive hypothesis it follows that each subtree rooted at r_i is executed by time $2e(r_i)$. Thus, T' will be executed by time $\tau + 2 \ \max\{e(r_i) \ : i = 1, \ldots, k\} + \tau + 1$. Since $e(r) = e(u_{\tau+1}) + \tau + 1$ and $e(u_{\tau+1}) \geq \max\{e(r_i) \ : i = 1, \ldots, k\}$, it follows that T will be executed by time $2e(r)$. ■

Next, using the modified PY method we show the following.

Theorem 1 *Let T be a tree dag with n tasks and root r. Let e be the function in the PY method for the PY model with parameter τ. Then, there exists a schedule for T on a linear array with m processors with makespan $T_{\max} \leq 4e(r) + \lceil n/(\tau+1) \rceil$, where $\lceil n/(2\tau+1) \rceil \leq m \leq \lceil n/(\tau+1) \rceil$. This schedule can be computed in polynomial time.*

Proof: Consider the schedule S for T produced by the modified PY method for the PY model with parameter τ. From Lemma 1 we know that S has no recomputation, and that each processor in the schedule S is assigned to execute a subtree of T with $\leq \tau + 1$ tasks.

We construct a tree T', which we call a *compressed tree*, with m_p nodes $v_1, v_2, \ldots, v_{m_p}$, where m_p is the number of processors used by the schedule S. Each node v_i of T' represents the subtree of tasks W_{v_i} assigned to each processor of the schedule S. Let r_{v_i} be the root task of W_{v_i}. There is an edge (v_i, v_j) in T' iff the successor of task r_{v_i} is in W_{v_j}. For each node v in T', let $\overline{W}_v$ be the subtree of T rooted at r_v and let n_v be the number of tasks in $\overline{W}_v$. Let u be a node of T' and let $u_1, u_2, \ldots, u_j$ be the children of u in T'. We order the children of u, from left to right, so that $4e(r_{u_1}) + \lceil n_{u_1}/(\tau+1) \rceil \leq 4e(r_{u_2}) + \lceil n_{u_2}/(\tau+1) \rceil \leq \ldots \leq 4e(r_{u_j}) + \lceil n_{u_j}|/(\tau+1) \rceil$, $j \leq \tau + 2$.

Consider a linear array with $\lceil n/(\tau+1) \rceil$ processors. Number its processors with consecutive positive integers so that the leftmost processor is numbered 1. Let Q_i denote the ith processor, $i = 1, 2, \ldots, \lceil n/(\tau+1) \rceil$. The *index* of a processor Q_i is equal to i.

We describe how the tasks of T are assigned to processors, and then we show how processors execute the tasks which have been assigned to them.

We assign the tasks of T to processors using a preorder traversal of T' as follows. To each processor we assign no more than $2\tau + 1$ tasks as described below. A processor is called *available* if it has been assigned less than $\tau + 1$ tasks. We perform a preorder traversal of T', and for each node v of T' we visit, we assign W_v to the lowest indexed available processor Q, and we say that v is assigned to Q. Since each W_v has no more than $\tau + 1$ tasks, no processor is assigned more than $2\tau + 1$ tasks. Observe that, if the height of v in T' is ≥ 1 then W_v has $\tau + 1$ tasks. Hence, no processor is assigned more than one node v of T' whose height in T' is ≥ 1.

Next, we prove that the above mapping has a "link-disjointness" property. (Preorder traversal, "link-disjointness", and level by level execution were used in [3] for a different compressed tree. The proof of link-disjointness is essentially as in [3].) Let v_1, v_2 be any two nodes of T' such that the subtrees T'_{v_1}, T'_{v_2} of T' rooted at v_1, v_2 are node-disjoint, and such that the preorder numbering of v_1 is less than that of v_2. Recall that $\overline{W}_{v_1}$ and $\overline{W}_{v_2}$ denote the subtrees of T rooted at r_{v_1} and r_{v_2} respectively. Note that $\overline{W}_{v_1}$ and $\overline{W}_{v_2}$ are the union of the subtrees of T which correspond to all the nodes of T'_{v_1} and T'_{v_2} respectively. The subtree $\overline{W}_{u_1}$ is assigned to a contiguous sub-array of processors $\mathcal{Q}_{v_1} = Q_{i_1}, \ldots, Q_{j_1}$, and the subtree $\overline{W}_{u_2}$ is assigned to a contiguous sub-array of processors $\mathcal{Q}_{v_2} = Q_{i_2}, \ldots, Q_{j_2}$, where $j_1 \leq i_2$. Intuitively, each subtree of T rooted at some task of T is assigned to a contiguous sub-array of processors of the linear array, and subtrees rooted at tasks of T not on the same path in T are mapped to link-disjoint sub-arrays of processors.

We describe how processors execute the tasks of T assigned to them. The description is given in terms of the nodes v of T', with the understanding that executing v by a processor means executing all tasks in W_v by that processor. Consider a processor Q. Initially,

processor Q executes all nodes $u_1, u_2, \ldots$, assigned to it, which have height 0 in T'. When Q finishes executing those nodes, it communicates the values of $r_{u_1}, r_{u_2}, \ldots$ to the processors that need them. If processor Q has been assigned no node of height > 0 in T', then Q will finish executing all its nodes by time $2\tau + 1$. However, if Q has been assigned a node v of T' whose height in T' is > 0, then Q will finish executing all its nodes of height 0 by time $\tau + 1$. Then, Q has to execute its node v of height > 0 in T'. Processor Q starts executing v after all the children of v have been executed, and Q received the values of all those children. Let t'_v be the time by which Q receives the values corresponding to the children of v. At time t'_v, processor Q starts executing v. Since W_v has $\tau + 1$ tasks, Q finishes executing all its tasks by time $t'_v + \tau + 1$. That is, Q follows a greedy strategy in executing the tasks assigned to it.

Next, we compute the time it takes to execute T by a linear array under the assignment and execution strategies just described. For each node v of T', let $\overline{t}_v$ be the time it takes for $\mathcal{Q}_v$ to execute $\overline{W}_v$. We claim the following:

Claim 1

$$\overline{t}_v \leq \begin{cases} 2\tau + 1, & \text{if } n_v \leq \tau + 1 \\ 4e(r_v) + \lceil n_v/(\tau+1) \rceil, & \text{otherwise} \end{cases}$$

Proof: We prove this claim by induction on the height k of $v \in T'$ in T'.

Basis. Let $v \in T'$ be a node of height 0 in T'. Then, because $\overline{W}_v$ has no more than $\tau + 1$ tasks, these tasks are executed by the processor to which they have been assigned by time $2\tau + 1$. Consider now a node v of T' with height 1 in T'. All the children of v will be executed by time $2\tau + 1$. Because v can have at most $\tau + 2$ children, we need to communicate at most $\tau + 2$ values to the processor assigned v. Using pipelining, this can be done in time no more than $\tau + 1 + \lceil n_v/(\tau+1) \rceil$. Because v has $\tau + 1$ tasks, we can execute v by time $4\tau + 3 + \lceil n_v/(\tau+1) \rceil$. Further, because $e(r_v) \geq \tau + 1$, it follows that T'_v will be executed by a linear array with no more than $\lceil n_v/(\tau+1) \rceil$ processors by time $4e(r_v) + \lceil n_v/(\tau+1) \rceil$.

Inductive hypothesis. Assume that the claim is true for all nodes v of T' whose height in T' is less than k, where $k \geq 1$.

Inductive step. Let v be a node of T' whose height in T' is $k \geq 2$. Let $u_1, u_2, \ldots, u_\lambda$ be all the children of v from left to right, $\lambda \leq \tau + 2$. Note that the tasks in W_v have been assigned to the leftmost processor of the contiguous linear sub-array $\mathcal{Q}_v$, and the tasks represented by the subtrees rooted at the children of v have been assigned to link-disjoint contiguous sub-arrays of processors of $\mathcal{Q}_v$. Further, the tasks of $\overline{W}_{u_\lambda}$ have been assigned to the rightmost part of $\mathcal{Q}_v$.

Because the height in T' of each child of v is less than k, we can apply the inductive hypothesis to them. Further, because $4e(r_{u_1}) + \lceil n_{u_1}/(\tau+1) \rceil \leq 4e(r_{u_2}) + \lceil n_{u_2}/(\tau+1) \rceil \leq \ldots \leq 4e(r_{u_\lambda}) + \lceil n_{u_\lambda}|/(\tau+1) \rceil$, all children of v will be executed by time $4e(r_{u_\lambda}) + \lceil n_{u_\lambda}/(\tau+1) \rceil$. After all children of v have been executed, we need to send the values of their roots, $r_{u_1}, r_{u_2}, \ldots, r_{u_\lambda}$,

to processor Q_{i_v}. Because W_v has $\tau + 1$ tasks, there are no more than $\tau + 2$ values that need to be communicated from various source processors to the processor assigned v. The distance from any source to the destination processor is no more than $\sum_{i=1}^{\lambda-1} \lceil n_{u_i}/(\tau+1) \rceil$. Using pipelining, all the values from the children of v can be communicated to the processor assigned v in time no more than $\tau + 1 + \sum_{i=1}^{\lambda-1} \lceil n_{u_i}/(\tau+1) \rceil$.

The processor assigned v will start executing W_v by time $4e(r_{u_\lambda})+\tau+1+\sum_{i=1}^{\lambda} \lceil n_{u_i}/(\tau+1) \rceil$. Because $\sum_{i=1}^{\lambda} \lceil n_{u_i}/(\tau+1) \rceil \leq \tau+1+\lceil n_v/(\tau+1) \rceil$, processor Q_{i_v} will start executing the tasks in W_v by time $4e(r_{u_\lambda}) + 2\tau + 2 + \lceil n_v/(\tau+1) \rceil$. Moreover, because each processor is assigned $\leq 2\tau + 1$ tasks and the processor assigned v finished executing all tasks except the tasks in W_v, it will finish executing W_v by time $4e(r_{u_\lambda}) + 3\tau + 3 + \lceil n_v/(\tau+1) \rceil$. From the definition of function e and from the construction of T', it follows that $e(r_v) \geq e(r_{u_l}) + \tau + 1$ for $l = 1, 2, \ldots, \lambda$, which implies that $4e(r_{u_\lambda})+3\tau+3+\lceil n_v/(\tau+1) \rceil \leq 4e(r_v)+\lceil n_v/(\tau+1) \rceil$. Therefore, all the tasks in the subtree of T rooted at r_v are executed by time $4e(r_v) + \lceil n_v/(\tau+1) \rceil$. ■

We conclude that T can be executed by a linear array with no more than $\lceil n/(\tau+1) \rceil$ processors and unit bandwidth links by time $4e(r)+\lceil n/(\tau+1) \rceil$. Further, since each processor of the linear array is assigned at most $2\tau + 1$ tasks, the linear array can have no less than $\lceil n/(2\tau+1) \rceil$ processors. Because the modified PY method works in polynomial time, it follows that we can find such a schedule for a tree dag on the linear array in polynomial time. ■

Further, since the makespan $T_{\max}$ of an optimal schedule for a tree dag T, with root r, on the PY model is such that $e(r) \leq T_{\max} \leq 2e(r)$, we have the following corollary to Theorem 1.

Corollary 1 *Let T be a tree dag with n tasks and height h. Let $T_{\max}$ be the makespan of an optimal schedule for T on the PY model with parameter τ. Then, there exists a polynomial time computable schedule for T on a linear array with $\Theta(n/\tau)$ processors and with $O(T_{\max} + n/\tau)$ makespan.*

Proof: Follows from Theorem 1 and the fact that $e(r) \leq T_{\max} \leq 2e(r)$, where r is the root of T. ■

Consider an optimal schedule S for a dag on the LIN–PY model, that uses m processors. Then, S is also an optimal schedule for that dag on the PY model with parameter $\tau = m - 1$. Using this fact, we have the following corollary to Theorem 1.

Corollary 2 *Let T be a tree dag with n tasks and height h. Let $T_{\max}$ be the makespan of an optimal schedule for T on the LIN–PY model, and let m be the number of processors used. Then, there exists a polynomial time computable schedule for T on a linear array with $O(n/m)$ processors and with $O(T_{\max})$ makespan.*

Proof: Follows from the facts that $\tau = m - 1$ and that $n/m \leq T_{\max}$. ■

Corollary 2 tells us that a linear array can simulate the LIN–PY model with constant slowdown.

4 Separating the Linear Array and the LIN–PY Model for Tree Dags

Since the linear array can simulate, for tree dags, the LIN–PY model with constant slowdown, we ask the question: Can the LIN–PY model simulate, for tree dags, the linear array with constant slowdown? Our investigation reveals that for binary tree dags with unit-time tasks the linear array is strictly more powerful than the LIN–PY model. More precisely, there exist infinitely many binary tree dags for which the LIN–PY model has optimal schedules whose makespan is not within a constant of the optimal makespan for those trees on the linear array. In other words, we show a separation theorem for the linear array and the LIN–PY model for the case of tree dags.

The above result is particularly interesting since the LIN–PY model has no notion of limited bandwidth. In $\tau = m - 1$ time units (where m is the number of processors), every processor can communicate with any other processor. This is generally not true for the linear array. For example, in a linear array with m processors, $\Omega(m^2)$ time units are needed to complete the following communication: processor i (when processors are numbered $1, 2, \ldots m$ from left to right) sends one value to each one of the processors $m-i$, for $i = 1, \ldots \lfloor m/2 \rfloor$. (So altogether, $\Omega(m^2)$ values are sent.) However, in the LIN–PY model, every communication step requires $\tau = m - 1$ time units, while in the linear array, processor i can send a value to processor $i + 1$ in one unit of time. Our results demonstrate that, for computing binary tree dags, the unlimited bandwidth of the LIN–PY model does not compensate for the over-estimation of communication delays.

Let T_0 be a complete binary tree dag of height $\lceil \log n/2 \rceil + 2\lceil \log \log n \rceil$. T_0 contains $\Theta(\sqrt{n} \log^2 n)$ tasks. We construct another binary tree dag $\overline{T}_0$ from T_0 by subdividing each edge of T_0 with $\lceil \sqrt{n}/\log^2 n \rceil$ nodes. $\overline{T}_0$ has $\Theta(n)$ nodes and height $h = \Theta(\sqrt{n}/(2 \log n) + 2\sqrt{n} \log \log n / \log^2 n)$. We prove that there exists a schedule for $\overline{T}_0$ on a linear array with $O(\sqrt{n})$ processors whose makespan is $O(\sqrt{n})$. Then, we show that the makespan of an optimal schedule for $\overline{T}_0$ on the LIN–PY model is $\Omega(\sqrt{n \log n}/\log \log n)$.

Lemma 2 *There exists a schedule for $\overline{T}_0$ on a linear array with $O(\sqrt{n})$ processors whose makespan is $O(\sqrt{n})$, i.e. optimal within a constant. This schedule can be found in polynomial time.*

Proof: We decompose $\overline{T}_0$ into two levels. Level 1 consists of the subtree rooted at the root

of $\overline{T}_0$ which contains the highest $\Theta(n/\log^2 n)$ ancestors of the root of $\overline{T}_0$. This subtree, called the *level–1 tree*, has height $\Theta(\sqrt{n}/\log n)$. Level 2 consists of the subtrees of $\overline{T}_0$ which belong to the forest formed when we remove the level–1 tree from $\overline{T}_0$. We call all these trees *level–2 trees*. There are $\Theta(\sqrt{n})$ level–2 trees and each level–2 tree has $\Theta(2^{2\log\log n}\sqrt{n}/\log^2 n) = \Theta(\sqrt{n})$ tasks and height $\Theta(\sqrt{n}\log\log n/\log^2 n)$.

Since each level–2 tree has $\Theta(\sqrt{n})$ tasks and there are $\Theta(\sqrt{n})$ level–2 trees, we can execute all of them in time $\Theta(\sqrt{n})$ using a linear array with $\Theta(\sqrt{n})$ processors.

Since the level–1 tree has $\Theta(n/\log^2 n)$ tasks and height $\Theta(\sqrt{n}/\log n)$, using the scheduling method by Ghosal *et al* [3], we can execute this tree on a linear array with $O(\sqrt{n}/\log n)$ processors in time $O((\sqrt{n}/\log n)\log n) = O(\sqrt{n})$.

However, before we start executing the level–1 tree we need to communicate all the values of the roots of the level–2 trees to the processors which will execute the successors of these roots in the level–1 tree. Because there are $\Theta(\sqrt{n})$ level–2 trees, and because the total number of processors used is $\Theta(\sqrt{n})$, using pipelining, we can communicate all these values to the corresponding processors executing tasks of the level–1 tree in time $\Theta(\sqrt{n})$. Therefore, $\overline{T}_0$ can be executed in time $O(\sqrt{n})$ by a linear array with $O(\sqrt{n})$ processors. Obviously, this schedule can be constructed in polynomial-time. ■

Next, we find a lower bound for the makespan of an optimal schedule for $\overline{T}_0$ on the LIN–PY model.

Lemma 3 *An optimal schedule for $\overline{T}_0$ on the LIN–PY model has $\Omega(\sqrt{n\log n}/\log\log n)$ makespan.*

Proof: Let S be an optimal schedule for $\overline{T}_0$ on the LIN–PY model and let τ be the value used as the interprocessor communication delay. By definition of a schedule for the LIN–PY model, τ is equal to the number of processors used in S minus 1. Further, the makespan $T_{\max}$ of S is no less than $e(r)$, where r is the root of $\overline{T}_0$, and e is the function in the PY method for the PY model with parameter τ. We consider two cases for τ.

Case 1: $\tau < \lceil\sqrt{n}/\sqrt{\log n}\rceil$. Suppose that $T_{\max} = o(\sqrt{n\log n}/\log\log n)$. The number of processors used in S is $m = \Omega(n/T_{\max}) = \omega(\sqrt{n}\log\log n/\sqrt{\log n})$. Because $\tau = m-1$, we have that $\tau = \omega(\sqrt{n}\log\log n/\sqrt{\log n})$, which is a contradiction. Thus, $T_{\max} = \Omega(\sqrt{n\log n}/\log\log n)$.

Case 2: $\tau \geq \lceil\sqrt{n}/\sqrt{\log n}\rceil$. Consider the function e in the PY method. We compute a lower bound on $e(r)$. Since $e(r) \geq \min\{n,\tau\}$, we assume, w.l.o.g, that $\tau \leq \lceil\sqrt{n\log n}\rceil$. Let $q = \lceil(\log\tau + \log(\log^2 n/\sqrt{n}))\sqrt{n}/\log^2 n\rceil$. We claim that for each task v of $\overline{T}_0$ of height iq, $e(v) \geq i\tau$. This claim can be easily proved by induction on i using the definition of function e. The key observation is that each task of height iq has $\geq \tau$ ancestors whose heights are between $(i-1)q$ and $iq-1$. Then, since the root r of $\overline{T}_0$ has some ancestor v of height $\Omega(h/q)$, and since

e is strictly monotonically increasing along a directed path in $\overline{T}_0$, it follows that $e(r) > e(v) = \Omega(\tau h/q)$. Further, since $h = \Theta(\sqrt{n}/(2\log n) + 2\sqrt{n}\log\log n/\log^2 n)$ and $\tau \leq \lceil\sqrt{n\log n}\rceil$, we have that $e(r) = \Omega(\tau \log n/\log\log n)$. Finally, since $\tau \geq \lceil\sqrt{n}/\sqrt{\log n}\rceil$, it follows that $e(r) = \Omega(\sqrt{n\log n}/\log\log n)$. Hence, the makespan of S is $\Omega(\sqrt{n\log n}/\log\log n)$. ■

We conclude that, even though we can find, in polynomial time, a schedule for $\overline{T}_0$ on a linear array with $O(\sqrt{n})$ processors and makespan $O(\sqrt{n})$, both optimal and processors–optimal within a constant, the optimal makespan for $\overline{T}_0$ on the LIN–PY model is $\Omega(\sqrt{n\log n}/\log\log n)$. This means that by taking the cost of all the communication steps to be τ, where τ equals the number of processors used minus 1, we over-estimate the total communication delay by more than a constant. In reality, there could be many communication steps which take $o(\tau)$ time. In addition, our result implies that, for tree dags, the unlimited communication bandwidth of the LIN–PY model does not compensate for this over-estimation. Therefore, even though the LIN–PY model, and consequently the PY model, are very useful approximation models for parallel machines, they do not always provide us with an accurate estimate of the time needed to execute a dag on an actual parallel machine.

Moreover, combining Corollary 2 and Lemmas 2 and 3, we get the following separation theorem between the LIN–PY model and the linear array for tree dags.

Theorem 2 *Let T_{OPT} be the makespan of an optimal makespan for a binary tree dag on a linear array, and let T_{LIN} be the makespan of an optimal schedule for that tree dag on the LIN–PY model. Then, there exists a constant $c > 0$ such that $T_{OPT} \leq c\ T_{LIN}$ for any binary tree dag, and there exists no constant $c' > 0$ such that, for all binary tree dags $T_{LIN} \leq c'\ T_{OPT}$. In other words, for binary tree dags, the linear array is strictly more powerful than the LIN–PY model.*

Proof: Follows from Corollary 2 and Lemmas 2 and 3. ■

5 Optimal within a Constant Schedules for Tree Dags on Linear Arrays

Using Theorem 1 and certain upper bounds on the e value of the root of a tree dag, we obtain polynomial time computable schedules for trees on linear arrays, that achieve makespan within a constant factor of optimal, for the following important ranges of the tree height h: $h \geq \sqrt{n}\log n$ and $h \leq n^{1/2-\epsilon}$ for any fixed $\epsilon > 0$. The schedule for $h \geq \sqrt{n}\log n$ has makespan $O(h)$, which is optimal within a constant, and uses $O(n\log n/h)$ processors, which is optimal within a factor of $\log n$. The schedule for $h \leq n^{1/2-\epsilon}$ has makespan $O(\sqrt{n})$, where the constant inside the Big Oh is proportional to $1/\epsilon$, and uses $O(\sqrt{n})$ processors, both being optimal within a constant factor when ϵ is fixed. These two results improve upon

the makespan of the schedules in Ghosal *et al.* [3] for those ranges of h, and at the same time also improve upon the time–processors product in [3] (for $h \leq n^{1/2-\epsilon}$), or maintain the same time–processors product as in [3] up to a constant (for $h \geq \sqrt{n}\log n$). We also prove that, for $h < \sqrt{n}$, our schedules achieve makespan $O(\sqrt{n}\log n/\log(\sqrt{n}/h))$ and use $O(\sqrt{n})$ processors. For the range $h \leq \sqrt{n}/\log^2 n$, we provide a polynomial time computable schedule on a linear array, that achieves makespan $O(\sqrt{n})$ using $O(\sqrt{n})$ processors (*i.e.* both optimal up to a constant), this time under an assumption that links have *unlimited bandwidth.*

Using Theorem 1 and a result of Thurimella and Yesha [11], we provide optimal within a constant schedules for any tree dag T with n tasks and height $h \geq \sqrt{n}\log n$ on a linear array with $O(n\log n/h)$ processors.

Corollary 3 *Let T be a tree dag with n tasks and height $h \geq \sqrt{n}\log n$. Then, there exists a polynomial time computable schedule for T on a linear array with $O(n\log n/h)$ processors and $O(h)$ makespan.*

Proof: We apply Theorem 1 with $\tau = \lceil h/\log n\rceil$. Let S be the schedule for T that we get by applying Theorem 1, and let $T_{\max}$ be the makespan of S. The number of processors m used in S is such that $\lceil n/(2\tau+1)\rceil \leq m \leq \lceil n/(\tau+1)\rceil$, which implies that $m = O(n\log n/h)$. Further, using a result in [11], we have that $e(r) \leq h + \tau\lceil\log n\rceil$, where r is the root of T. Because $T_{\max} \leq 4e(r) + \lceil n/(\tau+1)\rceil$, it follows that $T_{\max} \leq 4h + 4\tau\lceil\log n\rceil + \lceil n/(\tau+1)\rceil$. Because $\tau = \lceil h/\log n\rceil$ and $h \geq \sqrt{n}\log n$, we have that $T_{\max} = O(h)$. The makespan of S is optimal within a constant, and the number of processors used in S is optimal within a factor of $O(\log n)$. ■

This result improves by an $O(\log n)$ factor upon the makespan of the schedules in Ghosal *et al* [3] for this range of h, and at the same time maintains the same time–processors product within a constant. We note that Ghosal *et al* [3] use decomposition of the tree dag into paths, using node centroids recursively, in their schedule, while we follow a different approach.

For the case $h < \sqrt{n}$, we first provide an upper bound for the e value of the root of a tree dag. In particular, given a tree dag T with n tasks and height h, we show how to find in polynomial time a schedule for T on the PY model with parameter τ, $n > \tau > h$, with makespan $O(\tau\log(n/\tau)/\log(\tau/h))$ that uses $O(n/\tau)$ processors. [6] To this end, we are using the edge–centroid tree decomposition method.

Lemma 4 *Let T be a tree dag with n tasks and height h. Let τ be a positive integer such that $n > \tau > h$. Then, we can find, in polynomial time, a schedule for T on the PY model with parameter τ whose makespan is $O(\tau\log(n/\tau)/\log(\tau/h))$ and uses $O(n/\tau)$ processors.*

[6] Note that, if $\tau \geq n$ then there exists a trivial optimal schedule for T.

Proof: Consider the edge–centroid tree decomposition method. Using this method recursively, we can find a decomposition of a tree T with n nodes into $O(n/\tau)$ subtrees, each containing $O(\tau)$ tasks [2, 6], where $\tau > h$ is the parameter in the PY model.

We construct a sequence of trees using the aforementioned tree decomposition method. We denote the trees in this sequence by D_i. We define D_0 to be T, and $D_{-1} = \emptyset$. Consider a tree D_i, for some integer $i \geq 0$, which has n_i tasks and height no more than h. Tree D_{i+1} is constructed from D_i as follows. We decompose D_i into $O(n_i/\tau)$ subtrees $T_1, T_2, \ldots, T_{O(n_i/\tau)}$ each with $O(\tau)$ tasks. Let $r_1, r_2, \ldots, r_{O(n_i/\tau)}$ be the root of each such subtree. Tree D_{i+1} is the subtree of D_i induced by the tasks in D_i lying on a path from some r_j, $j = 1, 2, \ldots, O(n_i/\tau)$, to the root of D_i in D_i. It follows that D_{i+1} has $n_{i+1} = O((n_i/\tau)h)$ tasks and height h. Moreover, we can prove by induction on i that $n_i = O(n(h/\tau)^i)$. Since $\tau > h$, it follows that there exists a non-negative integer i such that $n_i = O(\tau)$. Let k be the smallest such integer.

Using the sequence $D_0, D_1, \ldots, D_k$ we describe a schedule for T. Consider a tree D_i in this sequence with $\omega(\tau)$ tasks. Consider the decomposition of D_i into $O(n_i/\tau)$ subtrees $T_1, T_2, \ldots, T_{O(n_i/\tau)}$ each with $O(\tau)$ tasks. For each subtree T_j, $j = 1, 2, \ldots, O(n_i/\tau)$, we schedule all the tasks in $T_j - D_{i+1}$ on one processor. Note that all the tasks in $T_j - D_{i+1}$ can be executed in $O(\tau)$ time, and that these tasks do not depend on tasks of D_i that belong to a different subtree. However, in order to keep the number of processors used $O(n_i/\tau)$ we pack subtrees to processors so that each processor is assigned $\Theta(\tau)$ tasks. This way we ensure that the number of processors used to execute $D_i - D_{i+1}$ is $O(n_i/\tau)$.

Unfortunately, because tasks in D_i depend on tasks in $D_{i-1} - D_i$, we need to communicate the values of these tasks in $D_{i-1} - D_i$ executed earlier to processors that need those values. Because in the PY model a processor can send an arbitrary number of values to any other processor in time τ, we can communicate all those values in τ time.

Therefore, we can execute all the tasks in $D_i - D_{i+1}$ on the PY model with parameter τ in $O(\tau)$ time using $O(n_i/\tau)$ processors. Note that if D_i has $O(\tau)$ tasks, then we can schedule D_i in $O(\tau)$ time using one processor.

Thus, we can schedule T in $O(k\tau)$ time using $O(n/\tau)$ processors. Because $n_k = O(n(h/\tau)^k)$ and k is the smallest integer such that $n_k = O(\tau)$, we have that $k = O(\log(n/\tau)/\log(\tau/h))$. Thus, we can schedule T in $O(\tau \log(n/\tau)/\log(\tau/h))$ time using $O(n/\tau)$ processors. ■

Next, we give an optimal within a constant schedule for a tree dag T with n tasks and height $h \leq n^{1/2-\epsilon}$, for any fixed $\epsilon > 0$, on a linear array with $O(\sqrt{n})$ processors. The makespan of this schedule is $O(\sqrt{n})$. So both the makespan and the number of processors are optimal within a constant factor. However, the constant inside the Big Oh for the makespan is proportional to $1/\epsilon$. This schedule is constructed using the modified PY method and the method in Theorem 1. Further, using Lemma 4, we find an upper bound for the makespan and number of processors of this schedule.

Theorem 3 *Let T be a binary tree dag with n tasks and height h, where $h \leq n^{1/2-\epsilon}$ for any*

fixed $\epsilon > 0$. Then, we can find, in polynomial time, a schedule for T on a linear array with $O(\sqrt{n})$ processors whose makespan is $O(\sqrt{n})$.

Proof: Let $\tau = \lceil \sqrt{n} \rceil$. From Lemma 4, it follows that there exists a schedule for T on the PY model with parameter τ, whose makespan is $O(\sqrt{n}\log(n)/\log(\sqrt{n}/h))$. This schedule uses $O(\sqrt{n})$ processors. Further, because $h < n^{1/2-\epsilon}$, the makespan for this schedule is $O(\sqrt{n})$. Moreover, this schedule implies that $e(r) = O(\sqrt{n})$, where r is the root of T and e is the function in the PY method.

Consider the schedule for T on the linear array produced by the method of Theorem 1, when $\tau = \lceil \sqrt{n} \rceil$. This schedule has makespan no more than $4e(r) + \lceil n/(\lceil \sqrt{n} \rceil + 1) \rceil$ and uses $O(\sqrt{n})$ processors. Since $e(r) = O(\sqrt{n})$, the makespan of this schedule is $O(\sqrt{n})$. For a fixed $\epsilon > 0$, this schedule is both optimal and processor optimal. ■

Note that, by extending Theorem 3, we can find, in polynomial time, a schedule for a tree dag, with n tasks and height $h < \sqrt{n}$, on a linear array with $O(\sqrt{n})$ processors so that its makespan is $O(\sqrt{n}\log n/\log(\sqrt{n}/h))$.

For the range $h \leq \sqrt{n}/\log^2 n$, we prove a result that sheds some light on the makespan complexity of trees on linear arrays. For this range of h, we provide a polynomial time computable schedule on a linear array, that achieves makespan $O(\sqrt{n})$ using $O(\sqrt{n})$ processors (*i.e.* both optimal up to a constant), this time under an assumption that links have *unlimited bandwidth.*

Theorem 4 *Let T be a tree dag with n tasks and height $h \leq \sqrt{n}/\log^2 n$. We can find, in polynomial time, an optimal within a constant schedule for T on a linear array with $O(\sqrt{n})$ processors under the assumption that links have unlimited bandwidth. The makespan of this schedule is $O(\sqrt{n})$.*

Proof: Consider the decomposition of a tree into $O(\sqrt{n})$ subtrees, each containing $O(\sqrt{n})$ tasks [2, 6]. Let T' be the compressed tree, which is constructed using this decomposition, that corresponds to T. Each node v of T' represents a subtree of tasks T_v of T rooted at a task r_v of T. This compressed tree T' has $O(\sqrt{n})$ nodes. Further, for each node v of T', the subtree T_v has $O(\sqrt{n})$ tasks of T.

Let $v_0, v_1, v_2, \ldots, v_k$, $k = O(\sqrt{n})$, be all the nodes of T', where v_0 is the root of T'. Let T'' be the subtree of T which consists of all tasks of T lying on the path from r_{v_i} to r_{v_0}, for $i = 1, 2, \ldots, k$. Since $k = O(\sqrt{n})$, this subtree T'' has $O(h\sqrt{n})$ tasks and height $O(h)$.

We map the nodes of T' to a linear array with $O(\sqrt{n})$ processors using a mapping of the nodes of T' as in Theorem 1 so that each processor gets $O(\sqrt{n})$ tasks. Because each processor is assigned $O(\sqrt{n})$ tasks of T, and because there are no dependencies among the tasks in $T - T''$ represented by different nodes of T', we can execute all the tasks in

$T - T''$ in time $O(\sqrt{n})$. Then, using the scheduling method by Ghosal *et al* [3], we schedule T'' on the same linear array of $O(\sqrt{n})$ processors. The makespan of this schedule for T'' is $O(\max\{h, \sqrt{h\sqrt{n}}\} \log \sqrt{h\sqrt{n}}) = O(\sqrt{n})$. However, because the tasks of T'' depend on $O(|T''|)$ values of $T - T''$, we must communicate all these values to the processors that need them. Because we assume that the linear array has links of unlimited bandwidth, this can be done, using pipelining, in $O(\sqrt{n})$ time, since the diameter of the linear array is $O(\sqrt{n})$. Therefore, the time to execute T by a linear array with $O(\sqrt{n})$ processors, assuming links of unlimited bandwidth, is $O(\sqrt{n})$. Further, this schedule can be found in polynomial time. ■

6 Conclusions

In this paper we give polynomial time algorithms to find schedules for binary tree dags on linear arrays, with makespan within a constant of optimal, for a range of the height h that excludes only the interval $n^{1/2-\epsilon} < h < n^{1/2} \log n$ (where ϵ can be chosen to be an arbitrarily small positive real number.) These schedules use our simulation result for the linear array, and the LIN–PY model.

Substantial effort is required to avoid link congestion in the schedules, since the linear array has links of unit bandwidth. Under an assumption of links of unlimited bandwidth, we provide an optimal within a constant factor schedule for the range $h \leq \sqrt{n}/\log^2 n$.

Using straightforward arguments (see [4]), some of our polynomial time scheduling algorithms can be extended to deal with the more general case, where the trees have degree bounded by some constant d, and the architecture is a k-dimensional mesh.

In the case of degree bounded by d, the lower bound on the makespan remains the same. When $h \geq \sqrt{n} \log n$, the upper bound on the makespan increases by a factor of d, due to the fact that a processor containing, say, a tasks may need up to da values communicated to it. When $h < n^{1/2}$, there is an additional factor of d due to using centroid decomposition.

In the case of meshes of dimension k, a straightforward argument in [4] shows that $\Omega(\max\{h, n^{1/(k+1)}\})$ is a lower bound on the makespan. Another straightforward argument in [4] shows how to simulate a linear array of m processors by a k-dimensional mesh of m processors, with a constant slowdown in communication. However, since the diameter of the k-dimensional mesh is $O(m^{1/k})$, the number of processors m can be increased to achieve more parallelism. Using these ideas, and our scheduling algorithm for the linear array, we obtain a polynomial time computable schedule on a k-dimensional mesh, for binary tree dags with $h \geq kn^{1/(k+1)} \log n$. The schedule has makespan $O(h)$ (*i.e.* within a constant of optimal), and uses $O(n \log n/h)$ processors. So the time–processors product is $O(n \log n)$ (*i.e.* within an $O(\log n)$ factor of optimal). This improves the makespan achieved in [4] while the same time–processors product is maintained, up to a constant factor. Similarly, when $h \leq kn^{1/(k+1)}/\log^2 n$, we can generalize our scheduling algorithm for the linear array

assuming links of unlimited bandwidth, to work for a k-dimensional mesh assuming links of unlimited bandwidth, producing a schedule with makespan $O(kn^{1/(k+1)})$, using $O(n^{k/(k+1)})$ processors. Both the makespan and number of processors are optimal up to a constant factor.

References

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

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

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

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

[5] H. Jung, L. Kirousis, and P. Spirakis. Lower bounds and efficient algorithms for multiprocessor scheduling of dags with communication delays. *Information and Computation*, 105, pages 94–104, 1993.

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

[7] H. T. Kung. Why systolic architectures. *IEEE Computer*, 15(1):37–46, January 1980.

[8] H. T. Kung. Systolic algorithms for the CMU WARP processor. *Proceedings of 7th International Conference on Pattern Recognition*, pages 570–577, July 1984.

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

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

[11] R. Thurimella and Y. Yesha. A scheduling principle for precedence graphs with communication delay. *Proceedings of the 1992 International Conference on Parallel Processing*, Vol. III, pages 229–236, 1992.