Upper and Lower Bounds on the Makespan of Schedules for Tree Dags on Linear Arrays

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

Abstract

We consider the problem of finding explicit tight upper and lower bounds on the makespan of schedules of tree dags on linear arrays, and the problem of polynomial time algorithms to find schedules that are optimal within a small constant.

We prove that $n+m+(m^2-5)/4$ is a lower bound on the time–processors product of a schedule for a tree dag with n tasks and height h on a linear array that uses m processors, and that $\max\{n^{1/2}, h\}$ is a lower bound on the makespan of those schedules.

We find, in polynomial time, a schedule for a complete binary tree dag with n unit execution time tasks on a linear array with $m < 4(n+1)^{1/2}$ processors whose makespan is $(1+o(1))(n/m+m/4)$, *i.e.* optimal within a factor of $1+o(1)$. The makespan of that schedule is $(1+o(1)n^{1/2}$ when $m = \lceil 2n^{1/2} \rceil$. Further, given a binary tree dag T with n tasks and height h, we find, in polynomial time, a schedule for T on a linear array, with $\leq 2\sqrt{n}+4$ processors whose makespan is $\leq 4\sqrt{n}+h+6$, *i.e.* optimal within a factor of $5+o(1)$. Moreover, there is no link contention in that schedule.

On the other hand, we prove that explicit lower and upper bounds on the makespan of optimal schedules of binary tree dags on linear arrays differ at least by a factor of $1+\sqrt{2}/2$. We also find, in polynomial time, schedules for bounded tree dags with n unit execution time tasks, degree d, and height $h \in o(n^{1/2}) \cup \omega(n^{1/2})$ on a linear array with $\leq 2n^{1/2}+2d$ processors and $(1+o(1))\max\{n^{1/2}, h\}$ makespan, *i.e.* optimal within a factor of $1+o(1)$, this time under the assumption of links with unlimited bandwidth.

Finally, we compute an improved upper bound on the makespan of an optimal schedule for a tree dag on the architecture independent model of Papadimitriou and Yannakakis [14], provided that its height not too large.

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

[1]Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250, U.S.A. E-mail: kalpakis@cs.umbc.edu and yayesha@cs.umbc.edu

[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.

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, unless we say otherwise, we assume that each link has unit bandwidth and unit propagation delay, all tree dags are of bounded degree, and that all tasks are unit execution time tasks.

In this paper, we consider the problem of finding explicit tight upper and lower bounds on the makespan of schedules of tree dags on linear arrays, and the problem of polynomial time algorithms to find schedules that are optimal within a small constant.

Papadimitriou and Ullman [13], Papadimitriou and Yannakakis [14], Jung *et al.* [8], and Aggarwal *et al.* [1] study the problem of finding efficient methods to execute given programs on parallel machines. Our model differs from the architecture independent model of Papadimitriou and Yannakakis [14], which we call the PY model, since in their model there is no notion of limited bandwidth and all communication steps take the same time τ. Further, our model differs from the model of Aggarwal *et al.* [1] since in their model pipelining is not allowed and all communication steps take the same time.

Ghosal *et al.* [6] give a polynomial time algorithm to find a schedule for a binary tree dag with n 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.) Ghosal *et al.* [7] extend their schedules to bounded degree tree dags with tasks of arbitrary positive integer execution times. Kalpakis and Yesha [9] give, for any fixed positive real ϵ, a polynomial time algorithm that finds a schedule for a tree dag with n 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 [9] also show that the makespan of an optimal schedule for tree dags with n tasks and height h for the PY model with interprocessor communication delay τ is $O(\tau \log(n/\tau)/\log(\tau/h))$. Further, they prove that, for tree dags, the linear array is strictly more powerful than the PY model when τ equals the diameter of that linear array. They show that there exist binary tree dags

with n tasks and height $o(n^{1/2})$ whose optimal schedule for the PY model 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 $O(n^{1/2})$.

Kalpakis and Yesha [10] improve upon the results in [9] by providing optimal within a constant explicit upper bounds on the makespan of schedules for tree dags on mesh arrays of processors, and polynomial time algorithms to find schedules with makespan matching these bounds. They find, in polynomial time, a (non–preemptive) schedule for a binary tree dag with n tasks and height h on a d–dimensional mesh array with m processors whose makespan is $O(n/m+n^{1/(d+1)}+h)$, *i.e.* optimal within a constant factor. We note here that for the case of binary tree dags and linear arrays the makespan of their schedules is $\leq 6n/m+5m+h+1$. Further, they extend these schedules to bounded degree forest dags with arbitrary positive integer execution time tasks and to meshes with the propagation delay of all the links an arbitrary positive integer.

Kalpakis and Yesha [10] 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. For any fixed positive real number ϵ, they provide polynomial time computable schedules for binary tree dags with n tasks and height $h \notin (g(n)n^{-\epsilon}, g(n)\log n)$ on any parallel architecture satisfying those conditions with optimal within a constant makespan $O(g(n)+h)$, where g is a function that depends only on that architecture. To construct these schedules they simulate for tree dags the PY model with a parallel architecture, where τ is $\geq$ the diameter of the machine used. Further, Kalpakis and Yesha [10] extend all their schedules to the case of bounded degree forest dags with tasks of arbitrary positive integer execution times and architectures with the propagation delay of all the links a given arbitrary positive integer.

We prove that $n+m+(m^2-5)/4$ is a lower bound on the time–processors product of a schedule for a tree dag with n tasks and height h on a linear array that uses m processors, and that $\max\{n^{1/2}, h\}$ is a lower bound on the makespan of those schedules. The key observation in deriving these bounds is that the makespan of a schedule is greater than or equal to the sum of the number of tasks assigned to a processor and the distance of that processor from the processor that has been assigned the root of that tree dag.

We provide an optimal within a factor of $1+o(1)$ schedule for a complete binary tree dag T with n tasks on a linear array with $m < 4(n+1)^{1/2}$ processors. In particular, we find, in polynomial time, a schedule for T on a linear array with m processors whose makespan is $(1+o(1))(n/m+m/4)$, *i.e.* optimal within a factor of $1+o(1)$. Taking $m = \lceil 2n^{1/2} \rceil$, we obtain a schedule for T on a linear array with $\lceil 2n^{1/2} \rceil$ processors whose makespan is $(1+o(1))n^{1/2}$, *i.e.* optimal within a factor of $1+o(1)$. The idea behind those schedules is to assign the root of T to a processor close to the middle processor of the linear array, while assigning to each processor a number of tasks that is proportional to its distance from the middle processor. However, additional effort is required in order to ensure low link contention.

Given a binary tree dag T with n tasks and height h, we find, in polynomial time, a schedule for T on a linear array, with $\leq 2\sqrt{n}+4$ processors whose makespan is $\leq 4\sqrt{n}+h+6$, *i.e.* optimal within a factor of $5 + o(1)$, which improves by a factor of $2(6/5)^{1/2}$ upon the schedules in [10]. Given an integer m, $5 \leq m \leq 4\lceil\sqrt{n}/2\rceil$, we find, in polynomial time, a schedule for T on a linear array with m processors whose makespan is $\leq 4n/(m - 4) + m + h + 2$, an improvement over the $6n/m + 5m + h + 1$ makespan schedules in [10]. Moreover, there is no link contention in both of these schedules. These schedules use the path–centroid decomposition method in [10] and a more careful assignment of the subtrees in that decomposition to processors.

We also prove that there is a gap between explicit lower and upper bounds on the makespan of schedules of binary tree dags on linear arrays. Specifically, we construct an infinite sequence of pairs of binary tree dags (T_1, T_2), each with n tasks and height h, where $2^{-1/2} - o(1) \leq h/n^{1/2} \leq 2^{-1/2} + o(1)$, and such that $T^{(1)}_{\max} \geq (1 + 2^{-1/2} - o(1))n^{1/2}$ and $T^{(2)}_{\max} = (1 + o(1))n^{1/2}$, where $T^{(j)}_{\max}$ is the makespan of an optimal schedule for T_j, $j = 1, 2$, on a linear array with $O(n^{1/2})$ processors.

Shedding some light on the role of the limited bandwidth of the links on the makespan, we find, in polynomial time, schedules for tree dags with n tasks, degree d, and height $h \in o(n^{1/2}) \cup \omega(n^{1/2})$ on a linear array with $\leq 2n^{1/2} + 2d$ processors and links of unlimited bandwidth, so that the makespan of these schedules is $(1 + o(1)) \max\{n^{1/2}, h\}$, *i.e.* optimal within a factor of $1 + o(1)$.

Finally, we compute an improved upper bound on the makespan of a schedule for a tree dag on the PY model, provided that its height not too large. Upper bounds on the makespan of schedules of tree dags on the PY model are used in [10] in order to estimate the makespan of some of the schedules for tree dags on parallel architectures given there. In particular, we show that the makespan T_{PY} of an optimal schedule of a tree dag with n tasks, degree d, and height h, on the PY model with τ an integer such that $(d + 1)h < \tau < n$ is $T_{PY} \leq (1 + \frac{1}{c})\frac{(\tau+1)\log(n/\tau)}{\log(\tau/(cdh))} + 4\tau + 2$, where c is an arbitrary real number such that $1 \leq c < \tau/((d + 1)h)$. By choosing c, so that the right hand side of the expression above is minimized, we improve upon the previous upper bound [9] which was with $c = 1$.

The rest of the paper is organized as follows. Preliminaries are in section 2. In section 3 we prove the lower bounds on the time–processors product and the makespan. In section 4 we present our optimal within a factor of $1 + o(1)$ makespan schedules for complete binary tree dags on linear arrays. Then, in section 5 we present the gap between explicit lower and upper bounds on the makespan of schedules of binary trees on linear arrays, and in section 6 we give our polynomial time computable optimal within a factor of $5 + o(1)$ makespan schedules of binary trees on linear arrays. In section 7 we give our polynomial time computable optimal within a factor of $1 + o(1)$ schedules for binary tree dags on linear arrays, this time under the assumption that links have unlimited bandwidth. Finally, in section 8 we present the improved upper bound on the makespan of schedules of trees on the PY model.

2 Preliminaries

2.1 Tree Dags, Linear Arrays, and Schedules

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 $w(u)$. For simplicity, we write $v \in T$ or $(u,v) \in T$ whenever v or (u,v) is a node or an edge in T respectively. A node v of T is called *successor* of a node u of T if $(u,v) \in T$, and node v is called a *predecessor* of node u if $(v,u) \in T$. A *leaf node* is a node with no predecessors. The level of a node $u \in T$ is equal to its distance from the root of T, while its height is equal to the height of the subtree of T rooted at u. Hereafter, unless we state otherwise, we assume bounded degree tree dags with tasks of unit execution times.

A linear array is modeled by an undirected connected graph that 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 (unless we state otherwise) constant bandwidth. The propagation delay of all the links is an arbitrary positive integer. Throughout this paper, unless we state otherwise, we assume that each link has unit bandwidth and unit propagation delay.

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 necessary for inverse tree dags [8]. 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 tree dag and a linear array, a schedule is called *optimal* if its makespan $T_{\max}$ is minimum among all possible schedules for that dag on that linear array. 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 whose makespan is t.

Given a bounded degree tree dag T and a linear array, our objective is to find a schedule for T 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 $T_{\max}$.

2.2 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, 12]. 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.

Given a tree dag T with n nodes and a positive integer $\beta \leq n$, Kalpakis and Yesha [10] develop another way to decompose a T into subtrees such that each subtree T_i in that

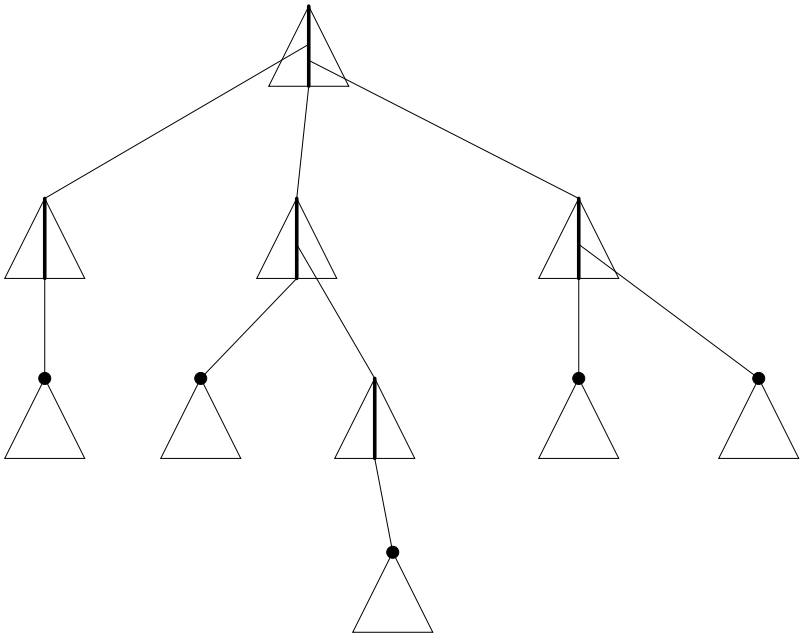

Figure 1: Path–centroid decomposition of a tree. Basic paths are denoted by thick lines (circles in case a basic path consists of a single node).

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, they 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.

To find find such a decomposition of T they combine the edge–centroid decomposition and the path decomposition methods. (See Appendix A for more details.) The resulting method is called the *path–centroid decomposition method*. See Fig. 1 for an example.

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: Omitted. See Kalpakis and Yesha [10] or Appendix A for details. ■

Given any decomposition of a bounded degree tree dag T into subtrees $T_1, T_2, \ldots, T_i, \ldots$, 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 *compressed tree* that corresponds to that decomposition of T. Moreover, for each subtree T_i in that decomposition we define a path in T_i, which we call the *basic path* that corresponds to T_i as follows. If T_i has a node whose predecessor(s) in T is not in T_i, then the basic path for T_i is the single path in T_i from that node to the root of T_i. Otherwise, basic path for T_i consists of the root of T_i only.

3 A Lower Bound on the Time-Processors Product of Schedules for Trees on Linear Arrays

We prove lower bounds on the time–processors product and the makespan of schedules for tree dags on linear arrays. These lower bounds are based on the observation that the makespan of a schedule is greater than or equal to the sum of the number of tasks assigned to a processor and the distance of that processor from the processor that has been assigned the root of that tree dag.

Theorem 1 *Let T be a tree dag of n unit execution time tasks, and let m be a positive integer. Then, the time–processors product of an optimal schedule for T on a linear array with m processors is*

$$mT_{\max} \geq n + m + \frac{m^2 - 5}{4}, \tag{1}$$

where $T_{\max}$ is the makespan of that schedule. In addition,

$$T_{\max} \geq \begin{cases} n, & \text{if } m = 1 \text{ or } n = 1 \\ n/m + m/4, & \text{if } m < \sqrt{4n-5} \\ \sqrt{4n-5}/2 + 1, & \text{otherwise.} \end{cases} \tag{2}$$

and $T_{\max} \geq \sqrt{n}$.

Proof: Number the processors of the linear array from left to right with consecutive positive integers so that the leftmost processor is numbered 1. Let k be the processor of the linear

array that has been assigned the root of T. Let x_i be the sum of the computation times of all the tasks that have been assigned to the ith processor, $1 \leq i \leq m$. We assume, without loss of generality, that $x_i > 0$ for all $1 \leq i \leq m$, *i.e.* each processor has been assigned at least one task of T.

Consider a processor i of the linear array. The value of the last task executed by that processor is needed in order to compute the root of T. Consequently, $T_{\max} \geq x_i + |i-k| + 1$ if $i \neq k$, and $T_{\max} \geq x_i + |i-k| + 1$ if $i = k$. Summing up for all $1 \leq i \leq m$, we get that

$$mT_{\max} \geq \sum_{i=1}^{m} x_i + \sum_{i=1}^{m} |i-k| + (m-1). \tag{3}$$

Since each task has to be executed by some processor, from (3) it follows that

$$mT_{\max} \geq n + \frac{k(k-1)}{2} + \frac{(m-k)(m-k+1)}{2} + m - 1 \geq n + m + (m^2-5)/4. \tag{4}$$

By dividing both sides of (4) by m and finding the minimum of the resulting right hand side, we obtain a lower bound on $T_{\max}$. Note that if $m = 1$ or $n = 1$ then $T_{\max} = n$. Suppose now that $n \geq 2$ and $m \geq 2$. From (4) we have that $T_{\max} \geq n/m + m/4 + 1 - 5/(4m)$, which implies that $T_{\max} \geq n/m + m/4$ if $m < \sqrt{4n-5}/2 + 1$ and that $T_{\max} \geq \sqrt{4n-5}/2 + 1$ otherwise. In all cases, $T_{\max} \geq \sqrt{n}$. ■

4 A $(1+o(1))n^{1/2}$ Makespan Schedule for Complete Binary Trees

We provide an optimal within a factor of $1 + o(1)$ schedule for a complete binary tree dag T on a given linear array. The idea is to assign the root of T to a processor close to the middle processor of the linear array, while assigning to each processor a number of tasks so that its distance from the middle processor plus its number of tasks is close to $n^{1/2}$. Further, tasks closer to the root are assigned to processors closer to the middle processor. However, since the number of values that need to routed towards the middle of the linear array can be large, additional effort is required in order to ensure low link contention. To achieve that, we assign at regularly spaced processors certain tasks whose role is to reduce the number of values that need to be routed towards the middle of the linear array. We assume, without loss of generality, using Theorem 1, that the number of processors m of a linear array used to schedule a tree dag with n unit time tasks is $< 4(n+1)^{1/2}$.

Theorem 2 *Let T be a complete binary tree dag with n unit execution time tasks and height h. Let m be a given positive integer $< 4(n+1)^{1/2}$. Then, we can find, in polynomial time, a*

schedule for T on a linear array with m processors whose makespan is $(1+o(1))(n/m+m/4)$, i.e. optimal within a factor of $1+o(1)$. In particular, when $m = \lceil 2n^{1/2} \rceil$, we obtain a schedule for T on a linear array with $\lceil 2n^{1/2} \rceil$ processors whose makespan is $(1+o(1))n^{1/2}$, i.e. optimal within a factor of $1 + o(1)$.

Proof: Let $\alpha = \lceil (h+3)/4 \rceil$, and let $x = 2^{\alpha+1} - 1$.

I. Partitioning the linear array.

Number the processors of the linear array with consecutive positive integers from left to right so that the leftmost processor is numbered 1. A *region* of the linear array is a contiguous sub–array of the linear array. Let $k = \lceil m/x \rceil$. Partition the linear array into k regions $R_1, R_2, \ldots, R_k$ such that region R_j, $1 \leq j \leq k-1$, consists of processors $(j-1)x+1$ through jx, while region R_k consists of the rightmost $m-(k-1)x$ processors of the linear array. The *index* of a region R_j is equal to j. The region of a processor i is R_j if processor i is contained in R_j. Let μ be the index of the region that contains processor $\lceil m/2 \rceil$. Region R_μ is called the *middle* region. We say that a region R_j is a *left region* if $1 \leq j \leq \mu$, and we say it is a *right region* otherwise. The *leader* of a region R_j is defined to be the highest numbered processor in R_j if R_j is a left region, otherwise it is defined to be the lowest numbered processor in R_j. The *middle leader* is defined to be the leader of R_μ, which is processor μx. The *distance* between two regions R_i and R_j is equal to $|i-j|$.

II. Scheduling the tree.

Our schedule for T consists of two phases. We execute all the tasks in $T - T'$ during phase 1, where T' is a certain subtree of T rooted at the root of T and defined below. Then, we execute all the tasks in T' in phase 2.

First, we define the *combinator* and the *envelope* of a subset of the tasks of T. Given a subset V of the tasks of T, we define the *combinator* $\mathrm{Comb}(V)$ to be the minimal fixed point solution of the following equation

$$\mathrm{Comb}(V) = \{\ u \mid u \in V \text{ or all predecessors of } u \text{ are in } \mathrm{Comb}(V)\ \}. \tag{5}$$

Note that $\mathrm{Comb}(V)$ can be computed by a straightforward iterative algorithm. Observe that, given the values of all the tasks in V, all other tasks in $\mathrm{Comb}(V)$ can be executed without ever needing a value of a task not in $\mathrm{Comb}(V)$. Further, since T is a complete binary tree dag, it follows that $|V| \leq |\mathrm{Comb}(V)| \leq 2|V| - 1$. We define the *envelope* $\mathrm{Env}(V)$ of V to be the set of tasks $u \in V$ such that the successor of u in T is not in V. Observe that, if all the tasks in V have been executed, then only the values of the tasks in the envelope $\mathrm{Env}(V)$ are needed to compute those tasks of T not in V. See Figure 2 for an example.

Fix a left to right order for the predecessors of each task of T. Let $v_1, v_2, \ldots, v_{2^{h-\alpha}}$, denote, in left-to-right contiguous order, the $2^{h-\alpha}$ tasks of T whose level is equal to $h - \alpha$.

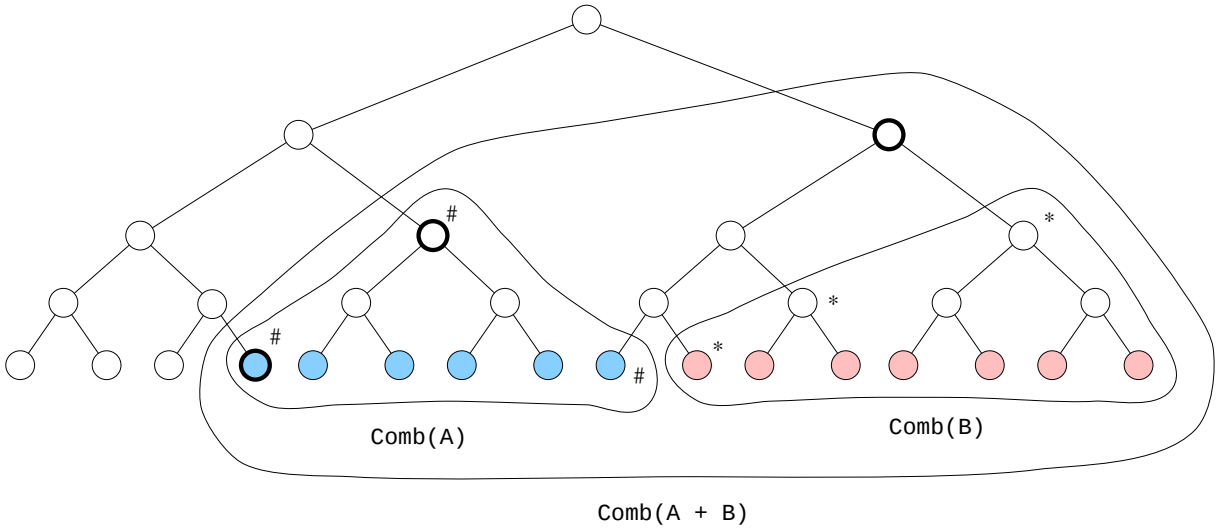

Figure 2: The combinators of two sets of nodes A and B. Shaded circles denote nodes in either A or B. Curves enclose the combinators of A, B, and $A \cup B$. Observe that $\text{Comb}(A \cup B) - (\text{Comb}(A) \cup \text{Comb}(B))$ is non-empty. Nodes in the envelope of $\text{Comb}(A)$ or $\text{Comb}(B)$ are marked with a # or $*$, respectively. Nodes in the envelope of $\text{Comb}(A \cup B)$ have thick outline.

Each task v_i is the root of a subtree T_i of T with height α and $x = 2^{\alpha+1} - 1$ tasks. Let $V_{i,j}$ be the set of tasks $v_i, v_{i+1}, v_{i+2}, \ldots, v_j$, $1 \leq i \leq j \leq 2^{h-\alpha}$. Note that, since T is a complete binary tree dag, $|V_{i,j}| \leq |\text{Comb}(V_{i,j})| \leq 2|V_{i,j}| - 1$ and that $|\text{Env}(\text{Comb}(V_{i,j}))| \leq 2(h - \alpha)$, $1 \leq i \leq j \leq 2^{h-\alpha}$.

Phase 1.

Task assignment. First, we assign the subtrees $T_1, T_2, \ldots, T_{2^{h-\alpha}}$ to the processors of the linear array as follows. We say that a processor in region R_j of the linear array, $1 \leq j \leq k$, is *available* if it has been assigned less than $\lceil n/(mx) + m/(4x) \rceil - |j - \mu - 1|$ of the subtrees T_i above. For $i = 1, 2, \ldots, 2^{h-\alpha}$, assign subtree T_i to the leftmost available processor. In other words, each processor is assigned a number of subtrees that decreases proportionally to the distance of its region from the middle region. Note also that we can assign at least $(n/2 + m/8)/2^{\alpha}$ subtrees to the m processors of the linear array. The set of tasks that consists of the roots of the subtrees assigned to the ith processor in region R_j forms a contiguous sub–sequence of the sequence $v_1, v_2, \ldots, v_{2^{h-\alpha}}$, which we denote by $Z_{i,j}$. Assign to processor i of region R_j the set of all the tasks in the combinator $\text{Comb}(Z_{i,j})$ of $Z_{i,j}$. Let $S_j = \text{Comb}(\cup_{i \in R_j} Z_{i,j}) - \cup_{i \in R_j} \text{Comb}(Z_{i,j})$. Assign to the leader processor of region R_j the set of tasks S_j. Note that S_j consists of all those tasks in the combinator $\text{Comb}(\cup_{i \in R_j} Z_{i,j})$ which have not already been assigned to some processor in region R_j.

The tasks of T that have not been assigned to any processor at this point induce a subtree T' of T. This subtree T' is rooted at the root of T and we call it the *top* subtree of

T. Observe that T' has $\leq 2^{h-\alpha}-1$ tasks and height $\leq h-\alpha-1$, and that if a task u of T' has a predecessor v not in T' then $v \in \cup_{j=1}^{k} \mathrm{Env}(S_j) = \mathrm{Env}(T-T')$.

Task execution. Processors execute the tasks in $T-T'$ assigned to them in greedy manner. Each processor p executes a task u assigned to it as soon as it becomes ready (break ties arbitrarily). If task u has a successor task u' and u' has not been assigned to p then the value of task u needs to routed to the processor that has been assigned u'. There are three cases to consider. If processor p is not the leader of its region then p routes the value of u to the leader of its region. Otherwise, if p is the leader of its region then p routes the value of u to the middle leader processor of the linear array. If p is the middle leader processor then it stores all the values that are routed to it; those values will be distributed during the second phase of our schedule. Routing is always done over a shortest path and link contention is resolved in FIFO order.

Let t_0 be the time to complete phase 1, that is the time at which the values of all the tasks in $\mathrm{Env}(T-T') = \cup_{j=1}^{k} \mathrm{Env}(S_j)$ are available to the middle leader processor.

Phase 2.

In phase 2, we execute the top subtree T'. Observe that each task in T' depends only on tasks in T' or in $\mathrm{Env}(T-T')$. In addition, the values of all the tasks in $\mathrm{Env}(T-T')$ are available to the middle leader processor by time t_0. It is easy to see that we can execute a complete binary tree dag with unit execution time tasks and height h on a linear array with $2^{\lceil h/2 \rceil}$ processors in time $\leq 4 \cdot 2^{\lceil h/2 \rceil}$. Consider a linear sub–array with $m' = \min\{m, 2^{\lceil (h-\alpha-1)/2 \rceil}\}$ processors, centered at the middle leader processor. Since $|\mathrm{Env}(T-T')| = |\cup_{j=1}^{k} \mathrm{Env}(S_j)| \leq 2(h-\alpha)\lceil m/x \rceil$, we can distribute the values of all the tasks in $\mathrm{Env}(T-T')$ from the middle leader processor of the array to each processor in that sub-array in time $2(h-\alpha)\lceil m/x \rceil + \lceil m'/2 \rceil$. Then, we can execute T' on that sub-array in time $4\lceil 2^{\lceil (h-\alpha-1)/2 \rceil}/m' \rceil 2^{\lceil (h-\alpha-1)/2 \rceil}$. Consequently, the makespan of our schedule for T is

$$T_{\max} \leq t_0 + 2(h-\alpha)\lceil m/x \rceil + \lceil m'/2 \rceil + 4\lceil 2^{\lceil (h-\alpha-1)/2 \rceil}/m' \rceil 2^{\lceil (h-\alpha-1)/2 \rceil}. \quad (6)$$

III. Bounding the makespan.

Bounding the time to complete Phase 1. The time t_0 to complete phase 1 is equal to the time to execute all the tasks in $T-T'$ and route the values of the tasks in $\mathrm{Env}(T-T')$ to the middle leader processor. Consider the assignment of the tasks in $T-T'$ to processors. Let $A_{i,j}$ denote the set of tasks in the union of the subtrees of T that are rooted at a task in $Z_{i,j}$. Processor i of region j is assigned the tasks in $A_{i,j} \cup \mathrm{Comb}(Z_{i,j}) \cup S_j$, if it is the leader of region R_j, and is assigned the tasks in $A_{i,j} \cup \mathrm{Comb}(Z_{i,j})$ otherwise.

First, we derive upper bounds on $|A_{i,j} \cup \mathrm{Comb}(Z_{i,j})|$ and $|S_j|$. Observe that, for each processor i in region R_j, $|Z_{i,j}| \leq \lceil n/(mx) + m/(4x) \rceil - |j-\mu-1|$. Since $|\mathrm{Comb}(Z_{i,j})| \leq 2|Z_{i,j}|-1$, it follows that

$$|A_{i,j} \cup \mathrm{Comb}(Z_{i,j})| \leq (x+2)(\lceil n/(mx) + m/(4x) \rceil - |j-\mu-1|). \quad (7)$$

Next, we find an upper bound on $|S_j|$. Consider the set of tasks S_j assigned to the leader of region R_j. Let T'_j be the subtree of T induced by the tasks in S_j. Observe that if a task $u \in T'_j$ has a predecessor v in T which is not in T'_j then v is in $\cup_{i \in R_j} \mathrm{Env}(\mathrm{Comb}(Z_{i,j}))$. Since T'_j is a binary tree, we conclude that $|S_j| \leq 3|\cup_{i \in R_j} \mathrm{Env}(\mathrm{Comb}(Z_{i,j}))|$. Further, since $|\cup_{i \in R_j} \mathrm{Env}(\mathrm{Comb}(Z_{i,j}))| \leq \sum_{i \in R_j} |\mathrm{Env}(\mathrm{Comb}(Z_{i,j}))|$ and $|\mathrm{Env}(\mathrm{Comb}(Z_{i,j}))| \leq 2(h-\alpha)$, it follows that

$$|S_j| \leq 6(h-\alpha)x. \tag{8}$$

Second, using the upper bounds computed above, we find an upper bound on the time that each processor takes to execute all the tasks assigned to it. Consider a processor i that is in a region R_j. Processor i executes all the tasks in $A_{i,j} \cup \mathrm{Comb}(Z_{i,j})$ assigned to it by time $(x+2)(\lceil n/(mx) + m/(4x) \rceil - |j-\mu-1|)$. If processor i is the leader of region R_j then it has also to execute the tasks in S_j. The tasks in S_j depend only on tasks in $S_j \cup (\cup_{i \in R_j} \mathrm{Env}(\mathrm{Comb}(Z_{i,j})))$. The values of all the tasks in $\cup_{i \in R_j} \mathrm{Env}(\mathrm{Comb}(Z_{i,j}))$ can be routed from the various processors in region R_j to its leader processor in time $\leq x + 2(h-\alpha)x$. Since $|S_j| \leq 6(h-\alpha)x$, it follows that the leader processor of region R_j finishes executing all the tasks in S_j by time

$$(x+2)(\lceil n/(mx) + m/(4x) \rceil - |j-\mu-1|) + x + 8(h-\alpha)x. \tag{9}$$

Third, we compute our upper bound on t_0, the time to complete phase 1. Since the leader processor of each region R_j routes the values of the tasks in $\mathrm{Env}(S_j)$ to the middle leader processor and $|\mathrm{Env}(S_j)| \leq 2(h-\alpha)$, it follows that the values of all the tasks in $\mathrm{Env}(S_j)$ are available to the middle leader processor by time t_j,

$$\begin{aligned} t_j \;\leq\; & (x+2)(\lceil n/(mx) + m/(4x) \rceil - |j-\mu-1|) + x + 8(h-\alpha)x + \\ & |j-\mu|x + 2(h-\alpha)\lceil m/x \rceil. \end{aligned} \tag{10}$$

Therefore, all the tasks in $T - T'$ are executed and the values of all the tasks in $\cup_{j=1}^{k} \mathrm{Env}(S_j)$ are available to the middle leader processor by time t_0,

$$t_0 \leq (x+2)\lceil n/(mx) + m/(4x) \rceil + 8(h-\alpha+1)x + 2(h-\alpha)\lceil m/x \rceil + 2. \tag{11}$$

Bounding the makespan of our schedule for T. Using (6) and (11), and after some algebra, we find that the makespan $T_{\max}$ of our schedule for T on a linear array with $m < 4(n+1)^{1/2}$ processors is

$$T_{\max} \leq n/m + m/4 + O(n^{3/8} + n^{3/4}/m) = n/m + m/4 + o(n/m). \tag{12}$$

Clearly, by taking $m = \lceil 2n^{1/2} \rceil$, we obtain a schedule for T on a linear array with $\lceil 2n^{1/2} \rceil$ processors and makespan $n^{1/2} + O(n^{3/8}) = (1+o(1))n^{1/2}$.

Next, we show how to derive (12). Recall that $\alpha = \lceil (h+3)/4 \rceil = \lceil (\log(n+1)+2)/4 \rceil$. Since $x = 2^{\alpha+1} - 1$, it follows that $(n+1)^{1/4} \leq x \leq 6(n+1)^{1/4}$. Then, since $m < 4(n+1)^{1/2}$, from equation (11) it follows that

$$t_0 \leq n/m + m/4 + 52(n+1)^{1/4}\log(n+1) + 2n^{3/4}/m + 4. \tag{13}$$

In addition,

$$2(h-\alpha)\lceil m/x \rceil \leq 9(n+1)^{1/4}\log(n+1). \tag{14}$$

Since $\lceil (h-\alpha-1)/2 \rceil \leq (3h+1)/8$, it follows that

$$3\lceil 2^{\lceil (h-\alpha-1)/2 \rceil}/m' \rceil 2^{\lceil (h-\alpha-1)/2 \rceil} \leq 3\lceil (n+1)^{3/8}/m' \rceil (n+1)^{3/8}. \tag{15}$$

Using (13), (14), and (15) in (6) we find that

$$\begin{aligned} T_{\max} \leq\ & n/m + m/4 + 52(n+1)^{1/4}\log(n+1) + 2n^{3/4}/m + 4 + \\ & 9(n+1)^{1/4}\log(n+1) + m'/2 + 1 + \\ & 3(n+1)^{6/8}/m' + 3(n+1)^{3/8}. \end{aligned} \tag{16}$$

Since $m' = \min\{m, 2^{\lceil (h-\alpha-1)/2 \rceil + 1}\}$, we have that

$$\min\{m, (n+1)^{3/8}/2\} \leq m' \leq \min\{m, 2(n+1)^{3/8}\}. \tag{17}$$

Then, from (16) and (17) it follows that

$$\begin{aligned} T_{\max} \leq\ & n/m + m/4 + 61(n+1)^{1/4}\log(n+1) + 2n^{3/4}/m + 5 + \\ & (n+1)^{3/8} + \max\{3(n+1)^{3/4}/m, 6(n+1)^{3/8}\}. \end{aligned} \tag{18}$$

from which it follows that

$$T_{\max} \leq n/m + m/4 + 7(n+1)^{3/8} + 8n^{3/4}/m + 61(n+1)^{1/4}\log(n+1) + 5. \tag{19}$$

Therefore, the makespan $T_{\max}$ of our schedule for T is given by (12). ■

We note here that Kalpakis and Yesha [11], using a simpler method, show a schedule for a complete binary tree dag with n unit execution tasks on a linear array with $\leq 2(n+1)^{1/2}$ processors whose makespan is $3(n+1)^{1/2}/2 + \log(n+1)/2 + 1$.

5 A Gap between Upper and Lower Bounds on the Makespan of Schedules of Trees on Linear Arrays

Its easy to see that the makespan $T_{\max}$ of a schedule for a tree dag with n tasks and height h on a linear array is at least h. Further, from Theorem 1, we have that $T_{\max} \geq n^{1/2}$. Moreover, there exist tree dags for which $T_{\max} = (1 + o(1))n^{1/2}$, namely a complete binary tree dag with n tasks, or $T_{\max} = h + 1$, namely a chain with $n = h + 1$ tasks. Consequently, $\max\{n^{1/2}, h\}$ is the best possible explicit lower bound on the makespan $T_{\max}$ of a schedule for a tree dag with n tasks and height h on a linear array, in the sense that any lower bound that depends only on n and h must be $\leq (1 + o(1)) \max\{n^{1/2}, h\}$. It is natural therefore to ask if $(1 + o(1)) \max\{n^{1/2}, h\}$ is an explicit upper bound on $T_{\max}$. In this section, we prove that any explicit lower and upper bounds on $T_{\max}$ must differ by a factor of at least $1+\sqrt{2}/2$. Hence, $(1 + o(1)) \max\{n^{1/2}, h\}$ can not be an explicit upper bound on $T_{\max}$.

Let $f_{lb}, f_{ub} : N \times N \rightarrow N$ be two functions such that $f_{lb}(n, h)$ and $f_{ub}(n, h)$ are a lower and an upper bound on the makespan of an optimal schedule of any binary tree dag T with n unit execution time tasks and height h on linear arrays with links of unit propagation delay and unit bandwidth. Without loss of generality, we only consider trees T for which $h \leq n^{1/2}$. We show that

$$\lim_{n\rightarrow\infty} \sup_{h\leq n^{1/2}} \frac{f_{lb}(n, h)}{n^{1/2}} = 1 \tag{20}$$

and

$$\lim_{n\rightarrow\infty} \inf_{h\leq n^{1/2}} \frac{f_{ub}(n, h)}{n^{1/2}} \geq 1 + \frac{\sqrt{2}}{2}, \tag{21}$$

where the inf and sup are taken over all binary tree dags with n tasks and height $h \leq n^{1/2}$. That is, there is a gap of at least $1+\sqrt{2}/2$ between upper and lower bounds on the makespan that depend only on n and h. Inequalities (20) and (21) follow from the next theorem.

Theorem 3 *There exists an infinite sequence of pairs of binary tree dags T_1 and T_2, each with n unit execution time tasks and height h, where $2^{-1/2} - o(1) \leq h/n^{1/2} \leq 2^{-1/2} + o(1)$, and such that $T^{(1)}_{\max} \geq (1+2^{-1/2}-o(1))n^{1/2}$ and $T^{(2)}_{\max} = (1+o(1))n^{1/2}$, where $T^{(j)}_{\max}$ is the makespan of an optimal schedule for T_j, $j = 1, 2$, on a linear array with links of unit propagation delay and unit bandwidth.*

Proof: Let i be a positive integer. Let $n = 2^{2i+1} + 2^i + 2i + 1$ and $h = 2^i + 2i + 1$. Note that $2^{-1/2} - o(1) \leq h/n^{1/2} \leq 2^{-1/2} + o(1)$. Let T_0 be a complete binary tree dag with $2^{2i+1} - 1$

tasks and height $2i$. Let r' be the root of T_0. Let P_1 be a path with $2^i + 1$ tasks, P_2 be a path with $2i + 1$ tasks, and P_3 be a path with $2^i + 2i + 2$ tasks.

For each i we construct two binary trees T_1 and T_2 as follows. Tree T_1 is constructed from P_1, P_2, and T_0 as follows: make the root of T_0 the only predecessor of the leaf of P_1 and make the root of P_2 a predecessor of the root of P_1. The tree T_2 is constructed from P_3 and T_0 as follows: make the root r' of T_0 to be a predecessor of the root of P_3. Clearly, both trees T_1 and T_2 have n tasks and height h.

First, consider an optimal schedule for T_1 on a linear array of processors. Let $T^{(1)}_{\max}$ be the makespan of that schedule. From Lemma 1 it follows that task r' is executed at time $\geq (2^{2i+1} - 1)^{1/2}$. Consequently, the root of T_1 is executed at time $\geq 2^i + (2^{2i+1} - 1)^{1/2} + 1$. Since

$$\lim_{i \to \infty} \frac{2^i + (2^{2i+1} - 1)^{1/2} + 1}{(2^{2i+1} + 2^i + 2i + 1)^{1/2}} = 1 + 2^{-1/2}, \tag{22}$$

it follows that $T^{(1)}_{\max} \geq (1 + 2^{-1/2} - o(1))n^{1/2}$.

Second, consider an optimal schedule for T_2 on a linear array of processors. Let $T^{(2)}_{\max}$ be the makespan of that schedule. From Theorem 2 it follows that T_0 can be executed by a linear array with $\lceil 2(2^{2i+1} - 1)^{1/2} \rceil$ processors in time $(1 + o(1))(2^{2i+1} - 1)^{1/2}$. By inserting an additional processor next to the processor of that linear array that has been assigned r' in the schedule provided by Theorem 2, it follows that $T^{(2)}_{\max} \leq \max\{(1 + o(1))(2^{2i+1} - 1)^{1/2}, 2^i + 2i + 1\} + O(1)$, while the number of processors used is $\lceil 2(2^{2h+1} - 1)^{1/2} \rceil + 1$. Since

$$\lim_{i \to \infty} \frac{\max\{(1 + o(1))(2^{2i+1} - 1)^{1/2}, 2^i + 2i + 1\} + O(1)}{(2^{2i+1} + 2^i + 2i + 1)^{1/2}} = 1 + o(1), \tag{23}$$

we conclude that $T^{(2)}_{\max} = (1 + o(1))n^{1/2}$. ■

Corollary 1 *Let $c1, c2, c3$ be positive real numbers. If $c_1 n^{1/2} + c_2 h + c_3 \max\{n^{1/2}, h\}$ is a lower bound on the makespan of an optimal schedule of a tree dag with n tasks and height h on a linear array then* $\max\{c_1 + c_2, c_1 + c_3, c_2 + c_3\} \leq 1$.

Proof: Follows from Theorem 3. ■

6 Optimal within a factor of $5 + o(1)$ Schedules for Binary Trees on Linear Arrays

Given a binary tree dag T with n tasks and height h, we can find, in polynomial time, a schedule for T on a linear array with $\leq 2\sqrt{n}+4$ processors whose makespan is $\leq 4\sqrt{n}+h+6$,

i.e. optimal within a factor of $5 + o(1)$. Further, given an integer $5 \leq m \leq 4\lceil\sqrt{n}/2\rceil$, we can find, in polynomial time, a schedule for T on a linear array with m processors whose makespan is $\leq 4n/(m-4) + m + h + 2$. Moreover, there is no link contention in any of these schedules. To accomplish that, we use the path–centroid decomposition method (see section 2.2). In particular, we do the following. First, given an integer $1 \leq B \leq n$ we decompose T into subtrees each with $\leq \lceil n/B \rceil$ tasks. Then, using that decomposition, we find, in polynomial time, a schedule for T on a linear array with m processors whose makespan is $\leq \lceil n/B \rceil + m + h + 1$, where m is the number of subtrees in that decomposition. Second, we show that the number of subtrees in any decomposition of T with parameter x, that is constructed using the edge–centroid method recursively, is no more than $\lceil 2n/x \rceil$. Since the number of subtrees in a path–centroid decomposition of T is at most twice the number of subtrees in a decomposition of T that is constructed using the edge–centroid method recursively, we conclude that $m \leq 4B$. Consequently, by choosing appropriate values for the parameter B we obtain the claimed results.

Lemma 2 *Let T be a bounded degree tree dag with n unit execution time tasks and height h. Let B be a positive integer $\leq n$. Let m be the number of subtrees in a path–centroid decomposition of T with parameter $\lceil n/B \rceil$. Then, we can find, in polynomial time, a schedule for T on a linear array with m processors and links of unit propagation delay and unit bandwidth whose makespan is $\leq \lceil n/B \rceil + m + h + 1$. Further, there is no link contention in this schedule.*

Proof: Consider the path-centroid compressed tree T_c with parameter $\lceil n/B \rceil$ (see section 2.2). Let m be the number of supernodes of T_c. Each supernode represents a subtree of T with $\leq \lceil n/B \rceil$ tasks. Our schedule for T is recursive and is based on this compressed tree T_c.

Task assignment. We assign the tasks of T to processors recursively using this compressed tree T_c. For brevity, hereafter, we say that we assign a supernode to a processor whenever all the tasks in the subtree of T represented by that supernode are assigned to that processor. We assign each supernode to a distinct processor.

First, we introduce the necessary definitions to describe our task assignment method. Let $L(u)$ denote the linear array to which all the supernodes in the subtree $T_c(u)$ of T_c rooted at a supernode u have been assigned. Let $l(u)$ be the distance between the processor in $L(u)$ that has been assigned supernode u and the farthest end–processor of $L(u)$. The end-processor of $L(u)$, that is closest to the processor that has been assigned u, is called the *closest–end processor* of $L(u)$. For each task v in the subtree of T represented by supernode u, let $l(v) = l(u)$, and let $t(v) = \lceil n/B \rceil + l(v) + h(v) + 1$, where $h(v)$ is the height of v in T. Further, let $t(u) = t(v')$, where v' is the root of the subtree represented by u. Intuitively, $t(v)$ is the time at which a task v on a basic path finishes executing.

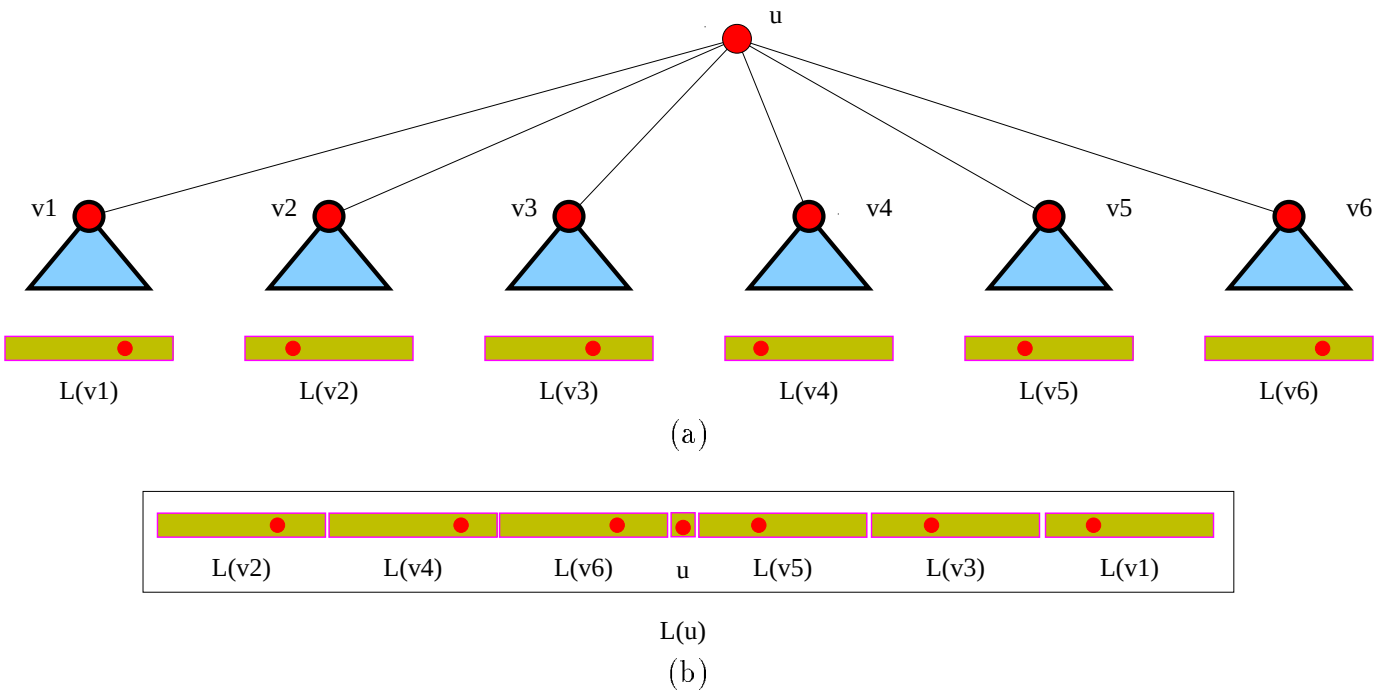

Figure 3: (a) The subtree $T_c(u)$ of a compressed tree T_c rooted at a supernode u. The predecessor supernodes $v_1, v_2, \ldots$ are sorted left-to-right in decreasing order of their $t(v_i)$ values. The linear arrays $L(v_i)$ to which $T_c(v_i)$ have been assigned are shown below each subtree. The processor of $L(v_i)$ which has been assigned v_i is also shown. (b) The linear array to which $T_c(u)$ is assigned.

Consider a subtree $T_c(u)$ of T_c that is rooted at a supernode u. If $T_c(u)$ has only one supernode, namely supernode u, then assign u to a linear array $L(u)$ with one processor. Otherwise, if $T_c(u)$ has more than one supernodes, do the following. Let $v_1, v_2, \ldots, v_j$ be the (direct) predecessors of u left–to–right in decreasing order of their $t(v_i)$ value. This ordering is needed to ensure that there will be no link congestion. Let L_0 be the linear array that results from the juxtaposition, of the linear arrays $L(v_{2i})$, $i = 1, 2, \ldots, \lfloor j/2 \rfloor$, reversed as necessary in order for the closest–end processor of $L(v_{2i})$ to be on the right. Let L_1 be the linear array that results from the juxtaposition, of the linear arrays $L(v_{2i+1})$, $i = \lfloor (j-1)/2 \rfloor, \ldots, 2, 1, 0$, reversed as necessary to ensure that the closest–end processor of $L(v_{2i+1})$ is on the left. Let $L(u)$ be a linear array with $|L_0| + |L_1| + 1 = |T_c(u)|$ processors. Identify the $|L_0|$ leftmost processors of $L(u)$ with those of L_0, assign the supernode u to the $|L_0|+1$ leftmost processor of $L(u)$, and identify the $|L_1|$ rightmost processors of $L(u)$ with those of L_1. See Fig 3 for an example. Requiring that the closest-end processors of the $L(v_i)$'s are towards the processor assigned u is necessary in order to keep the delays due to distances traveled small.

Task execution and routing regime. Consider a processor p and a task v that has been assigned to p. If task v is neither on a basic path nor the root of subtree represented by

a supernode of T_c, then processor p executes task v in a greedy manner, *i.e.* it executes that task as soon as it becomes ready (break ties arbitrarily). Otherwise, processor p starts executing task v at time $t(v)-1$. In both cases, task v is executed by time $t(v)$. The routing of the value of a task v from a processor p to a processor p' is done over the unique path in the linear array from p to p'.

In the remainder of this proof, we show that the task execution and routing regime provide us with a valid schedule for T.

First, we show that there is no link congestion. Observe that the only values that ever need to be routed are the values of tasks that are roots of subtrees corresponding to supernodes in T_c. Let v_1, v_2 be two tasks whose values need to be routed and such that the paths used to route the values of these tasks have at least one link in common. Let u_1, u_2 be the supernodes that correspond to v_1, v_2, respectively. Let u_0 be the least common ancestor supernode of u_1, u_2 in T_c. Let p_i be the processor that has been assigned supernode u_i, $i = 0, 1, 2$. Suppose, without loss of generality, that p_1 is farthest from p_0 than p_2 is, *i.e.* that p_2 is between p_1 and p_0. Then, it follows from the task assignment method that u_1 is a (direct) predecessor of u_0. Let u_2' be the (direct) predecessor of u_0 that is also an ancestor of u_2 ($u_2' = u_2$ if u_2 is a direct predecessor of u_0). From the task assignment method, since p_2 is between p_1 and p_0, and since the predecessors of u_0 are sorted in decreasing order of their t values, it follows that $t(u_1) \geq t(u_2') \geq t(u_2)$. Hence, $t(v_1) \geq t(v_2)$. Consequently, the values of tasks v_1 and v_2 can not compete for using the same link at the same time. Thus, there is no link congestion.

Second, we prove that the task execution and routing regime provide us with a valid schedule for T. All the tasks of T, that are neither roots of the subtrees represented by supernodes of T_c nor on basic paths of the path–centroid decomposition of T, are executed by time $\lceil n/B \rceil$. The remaining tasks of T induce a subtree T' of T rooted at the root of T. Note that each processor can have at most one ready task after time $\lceil n/B \rceil$. We prove, by induction on $h(v)$, that each task v in T' can indeed be executed by time $t(v) = \lceil n/B \rceil + l(v) + h(v) + 1$.

Basis: The claim is trivially true for each task v in T' of height $h(v) = 0$.

Inductive hypothesis: Suppose that each task v of T' with height $h(v) < k$ is executed by time $t(v)$, for any integer $k \geq 1$.

Inductive step: Consider a task v of T' of height $h(v) = k$. Let p_v be the processor that has been assigned task v. Let u be a predecessor of v in T and let p_u be the processor that has been assigned task u. We show that the value of task u is available to processor p_v by time $t(v) - 1$. If task u is not in T' then $p_u = p_v$ and task u has been executed by processor p_v by time $\lceil n/B \rceil$. If u is in T' and $p_u = p_v$, then by the inductive hypothesis task u is executed by time $t(u) = \lceil n/B \rceil + h(u) + l(u) + 1$. Hence, in both these cases, since $h(v) \geq h(u) + 1$, the value of task u is available to processor p_v by time $t(v) - 1$. Otherwise, task u is in T' and $p_u \neq p_v$. By the inductive hypothesis, task v is executed

by processor p_u at time $t(u) = \lceil n/B \rceil + l(u) + h(u) + 1$. Since there is no link contention, the value of task u is available to processor p_v at time $t(u) + d(u,v)$, where $d(u,v)$ is the distance between processors p_u and p_v. Since the closest-end processor of the linear array used to schedule the subtree of T rooted at u is towards processor p_v, we conclude that $l(v) \geq l(u) + d(u,v)$. Therefore, the value of task u is available to processor p_v by time $\lceil n/B \rceil + l(v) + h(u) + 1 \leq t(v) - 1$. Therefore, task v becomes ready by time $t(v) - 1$, which implies that task v finishes execution by processor p_v at time $t(v)$.

Thus, the root r of T is executed by time $\lceil n/B \rceil + l(r) + h + 1$. Further, since $l(r) + s(r) \leq m$, the value of the root r is available to an end processor of the linear array by time $\lceil n/B \rceil + m + h + 1$. ■

In order to utilize Lemma 2 above, we need to have a bound on the number m of subtrees in a path–centroid decomposition of T with parameter $\lceil n/B \rceil$. Since m is at most twice the number of subtrees m' in a decomposition of T that is constructed using the edge–centroid method recursively and the same parameter $\lceil n/B \rceil$, we find an upper bound on m'.

Lemma 3 *Let T be a tree with n nodes and degree d, and let x be a positive integer $\leq n$. Then, using the edge–centroid decomposition method recursively, we can decompose T into $\leq \lceil dn/x \rceil$ subtrees each with $\leq x$ nodes.*

Proof: Assume, without loss of generality, that $d \geq 2$.

Consider a decomposition $\mathcal{D}$ of T into subtrees each with $\leq x$ nodes using the edge-centroid method recursively. To this decomposition of T, there corresponds a node–weighted partition tree T_p as follows. If T has $\leq x$ nodes then T_p has only one node whose weight is equal to the number of nodes in T. If T has more than x nodes, then let T_1, T_2 be the two trees into which T is partitioned by the edge–centroid decomposition method. Then, T_p has as a root a new node, whose weight is equal to the number of nodes of T, and with left and right children the partition trees that correspond to decompositions of T_1 and T_2 into subtrees of size $\leq x$ using the edge-centroid method recursively. Hence, for each subtree in $\mathcal{D}$, there is a corresponding leaf of T_p, which has node weight equal to the number of nodes in that subtree. Let $w(u)$ denote the weight of a node u of T_p. Note that for any two nodes u, v of T_p such that u is the parent of v we have that $1/(d+1) \leq w(v)/w(u) \leq d/(d+1)$.

Since the number of subtrees $\mathcal{D}$ is equal to the number of leaf nodes of T_p, we proceed to find an upper bound on the number of leaves of T_p.

Let A be the set of nodes v of T_p such that v has a child that is a leaf but none of the children of the parent of v in T_p is a leaf. For each $v \in A$ we have that $x < w(v) \leq (d+1)x$. Moreover, for each leaf u of T_p there exists a unique node v in A such that u is in the subtree of T_p rooted at v. Thus, to count the leaves of T_p we count the leaves of the subtrees of T_p rooted at nodes in A. Partition the nodes of A into $m = \lceil \log(d+1)/(\log(d+1) - \log d) \rceil$

sets $A_1, A_2, \ldots, A_m$ based on their weight such that

$$A_i = \{v \in A \ : \ ((d+1)/d)^{i-1} < w(v)/x \leq ((d+1)/d)^i\}, \quad \text{for } i = 1, 2, \ldots, m. \tag{24}$$

Next, we prove, by induction on i, that the number of leaves in any subtree of T_p that is rooted at a node in A_i is $\leq i+1$, for $i = 1, 2, \ldots, m$.

Basis: Consider a node $v \in A_1$. The weight $w(v)$ of v is such that $1 < w(v)/x \leq (d+1)/d$. Then, both children of v in T_p have weight $\leq x$, which implies that both are leaves of T_p.

Inductive hypothesis: Suppose that, for any integer $2 \leq k < m$, any subtree of T_p rooted at a node in A_i has at most $i+1$ leaves, for any positive integer $i < k$.

Inductive step: Consider a subtree of T_p that is rooted at a node $v \in A_k$. Let v_1, v_2 be the two children of v in T_p. Since $v \in A$, at least one of its children, say v_1, is a leaf of T_p. Since $w(v_2)/w(v) \leq d/(d+1)$ and $w(v)/x \leq ((d+1)/d)^k$, it follows that $w(v_2)/x \leq ((d+1)/d)^{(k-1)}$. Hence, $v_2 \in A_i$ for some positive integer $i \leq k-1$. By the inductive hypothesis, the number of leaves in the subtree of T_p rooted at v_2 is $\leq i+1 \leq k$. Consequently, the number of leaves in the subtree of T_p rooted at a node in A_k is $\leq k+1$.

Finally, we bound the number of subtrees in $\mathcal{D}$. The number of leaves in T_p is $\leq \sum_{i=1}^{m}(i+1)|A_i|$. It can be shown, by induction on i, that $i+1 \leq d((d+1)/d)^{i-1}$ for $i \geq 1$. Moreover, since each node in A_i has weight greater than $((d+1)/d)^{i-1}x$, and since $\sum_{v \in A} w(v) = n$, it follows that

$$\sum_{i=1}^{m}(i+1)|A_i| \leq d\sum_{i=1}^{m}|A_i|\left(\frac{d+1}{d}\right)^{i-1} \leq \frac{dn}{x}. \tag{25}$$

Consequently, the number of leaves in T_p, which is equal to $|\mathcal{D}|$, is $\leq \lceil dn/x \rceil$. ■

Combining Lemmas 2 and 3, we show the following.

Theorem 4 *Let T be a bounded degree tree dag with n unit execution time tasks, height h, and degree d. Let B be a positive integer $\leq n$. Then, we can find, in polynomial time, a schedule for T in a linear array with at most $2dB$ processors and links of unit propagation delay and unit bandwidth whose makespan is $\leq \lceil n/B \rceil + 2dB + h + 1$. Further, there is no link contention in this schedule.*

Proof: Consider a path–centroid decomposition of T with parameter $\lceil n/B \rceil$. This decomposition is obtained by a decomposition of T into subtrees each with $\leq \lceil n/B \rceil$ nodes, that is constructed using the edge–centroid method recursively. From Lemma 3 it follows that the number of subtrees in this later decomposition is $\leq \lceil dn/\lceil n/B \rceil \rceil \leq dB$, which implies that the number of subtrees in that path–centroid decomposition of T is $\leq 2dB$. The theorem now follows from Lemma 2. ■

Corollary 2 *Let T be a binary tree dag with n unit execution time tasks and height h. Then, we can find, in polynomial time, a schedule for T on a linear array, with $\leq 2\sqrt{n}+4$ processors and links of unit propagation delay, whose makespan is $\leq 4\sqrt{n}+h+6$, i.e. optimal within $5+o(1)$. Further, given an integer $m \geq 5$, if $m \leq 4\lceil\sqrt{n}/2\rceil$ then we can find, in polynomial time, a schedule for T on a linear array with m processors whose makespan is $\leq 4n/(m-4)+m+h+2$. In addition, if $h=\omega(\sqrt{n})$ then we can find, in polynomial time, a schedule for T on a linear array with $\leq 4f(n)n/h+4$ processors whose makespan is $\leq (1+1/f(n))h+4f(n)n/h+6$, i.e. optimal within $1+o(1)$ and processors optimal within $4f(n)$, where $f(n)$ is a function in $\omega(1)\cap o(h/\sqrt{n})$. Moreover, there is no link contention in any of these schedules.*

Proof: Use Theorem 4. Take $B=\lceil\sqrt{n}/2\rceil$, $B=\lfloor m/4\rfloor$, and $B=\lceil f(n)n/h\rceil$, in the first, second, and third case respectively. ■

Lemmas 2 and 3, Theorem 4, and Corollary 2 can be extended to the case where tasks have arbitrary positive integer execution times and each link of the linear array has propagation delay an arbitrary positive integer r_0 (the same for each link). They can also be extended to forest dags (the idea is to pack small trees together, and execute large trees using the schedules in this section).

7 Optimal within a factor of $1+o(1)$ Schedules for Trees on Linear Arrays with Unlimited Bandwidth Links

We show how to find schedules for bounded degree d tree dags with n unit execution time tasks and height $h \in o(n^{1/2}) \cup \omega(n^{1/2})$ on a linear array with $\leq 2n^{1/2}+2d$ processors and links of unlimited bandwidth, so that the makespan $T_{\max}$ of these schedules is $T_{\max} = (1+o(1))\max\{n^{1/2}, h\}$, *i.e.* optimal within a factor of $1+o(1)$.

Lemma 4 *Let T be a bounded degree tree dag T with n unit execution time tasks and height $h=o(n^{1/2})$. Then, we can find, in polynomial time, a schedule for T on a linear array with $\lceil 2n^{1/2}\rceil$ processors and links with unlimited bandwidth so that the makespan of that schedule is $T_{\max}=(1+o(1))n^{1/2}$.*

Proof: Since $h=o(n^{1/2})$, there exist positive real numbers ϵ and δ such that $\epsilon<\delta<1/2$ and $h \leq n^{\epsilon}$. Using the edge–centroid decomposition method, partition T into k subtrees $T_1, T_2, T_3, \ldots, T_k$, such that each subtree has between $\lfloor\lceil n^{\delta}\rceil/3\rfloor$ and $\lceil n^{\delta}\rceil$ tasks. Observe that $\lceil n^{1-\delta}\rceil \leq k \leq \lceil dn^{1-\delta}\rceil$. Let T' be the subtree of T that consists of all those tasks of T that are on a path from the root of T_i to the root of T, for each $i=1,2,\ldots,k$. Since $h \leq n^{\epsilon}$ and

$\epsilon < \delta < 1/2$, it follows that T' has at most $\lceil dn^{1-\delta} \rceil h = o(n)$ tasks. Further, note that T' has at least $n^{1-\delta}/2 > n^{1/2}/2$ tasks.

Our schedule for T consists of two phases. In the first phase we schedule and execute all the tasks in $T - T'$. In the second phase, using Theorem 4, we schedule and execute all the tasks in T'.

Phase I.
Our aim is to assign to each processor a number of tasks that decreases proportionally to its distance from the middle of the linear array, while the middle processor gets about $n^{1/2}$ tasks. Processors will execute the tasks assigned to them in a greedy manner, and they will route the values of those tasks that have their successor in T' to the middle processor.

Consider a linear array with $\lceil 2n^{1/2} \rceil$ processors. Number its processors with consecutive positive integers from left to right so that the leftmost processor is numbered 1. Processor j is *available* if it has been assigned $< \lceil n^{1/2} \rceil - |\lceil n^{1/2} \rceil - j| + 1$ tasks.

For $i = 1, 2, \ldots, k$, assign all the tasks in $T_i - T'$ to the leftmost available processor. Let m be the rightmost processor that has been assigned at least one task. Each processor $j < m$ gets between $\lceil n^{1/2} \rceil - |\lceil n^{1/2} \rceil - j| + 1$ and $\lceil n^{\delta} \rceil + \lceil n^{1/2} \rceil - |\lceil n^{1/2} \rceil - j| + 1$ tasks. Note that, since

$$\sum_{j=1}^{\lceil 2n^{1/2} \rceil} \left(\lceil n^{1/2} \rceil - |\lceil n^{1/2} \rceil - j| + 1 \right) \geq n, \tag{26}$$

there are enough processors to assign all the tasks in $T - T'$.

Processors execute the tasks in $T - T'$ in greedy manner, *i.e.* a processor executes a task that has been assigned to it as soon as that task becomes ready (break ties arbitrarily). Whenever a processor j executes a task v that has been assigned to it and whose successor task is in T', processor j routes the value of task v to processor $\lceil n^{1/2} \rceil$ over the single path that connects these two processors. Because links have unlimited bandwidth, there is no link contention.

Consider processor j, $1 \leq j \leq m$. All the tasks that have been assigned to it have all their ancestors also assigned to the same processor j. Because processor j has been assigned $\leq \lceil n^{\delta} \rceil + \lceil n^{1/2} \rceil - |\lceil n^{1/2} \rceil - j| + 1$ tasks, processor j finishes executing all the tasks that have been assigned to it by time $\lceil n^{\delta} \rceil + \lceil n^{1/2} \rceil - |\lceil n^{1/2} \rceil - j| + 1$. Moreover, since we assume links of unlimited bandwidth, the values of all the tasks executed by processor j, which have their successors in T', become available to processor $\lceil n^{1/2} \rceil$ by time $\lceil n^{\delta} \rceil + \lceil n^{1/2} \rceil + 1 \leq n^{1/2} + n^{\delta} + 3$.

Phase II.
Using Theorem 4, and since T' has $o(n)$ tasks and height $o(n^{1/2})$, we can schedule T' on a linear array with $o(n^{1/2})$ processors in time $o(n^{1/2})$. Note that, since we assume links of

unlimited bandwidth, the values of those tasks in $T - T'$ that are needed to execute T' can be redistributed from the middle processor to the processors that need them in time $o(n^{1/2})$.

Consequently, we conclude that the time to execute all the tasks of T is $n^{1/2} + n^{\delta} + o(n^{1/2})$. Since $\delta < 1/2$, it follows that the makespan of our schedule for T is $(1 + o(1))n^{1/2}$, while it uses $\leq \lceil 2n^{1/2} \rceil$ processors. ■

Using Theorem 4 and Lemma 4 we have the following.

Theorem 5 *Let T be a bounded degree tree dag T with n unit execution time tasks and height $h = o(n^{1/2}) \cup \omega(n^{1/2})$. Let d be the degree of T. Then, we can find, in polynomial time, a schedule for T on a linear array with $\leq 2n^{1/2} + 2d$ processors and links with unlimited bandwidth so that the makespan of that schedule is $T_{\max} = (1 + o(1)) \max\{n^{1/2}, h\}$, i.e. optimal within a factor of $1 + o(1)$.*

Proof: When $h = o(n^{1/2})$, the theorem follows from Lemma 4. When $h = \omega(n^{1/2})$, the theorem follows from Theorem 4 by taking $B = \lceil n^{1/2}/d \rceil$. ■

8 An Improved Upper Bound on the Makespan of Schedules for Tree dags on the PY Model

We compute an improved upper bound on the makespan of a schedule for a tree dag on the PY model, provided that its height not too large. The PY model [14] 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 $t + \tau$, while there is an unlimited number of identical processors available. Upper bounds on the makespan of schedules of tree dags on the PY model are used [10] in order to estimate the makespan of some of the schedules for tree dags on parallel architectures given there. We show the following.

Lemma 5 *Let T be a tree dag with n unit execution time tasks, degree d, and height h. Let τ be a positive integer such that $n > \tau > (d + 1)h$. Then, we can find, in polynomial time, a schedule for T on the PY model with parameter τ whose makespan is*

$$T_{PY} \leq (1 + \frac{1}{c}) \frac{(\tau + 1) \log(n/\tau)}{\log(\tau/(cdh))} + 4\tau + 2, \tag{27}$$

using $\leq \lceil 2cn/\tau \rceil$ processors, where c is an arbitrary real number such that $1 \leq c < \tau/((d + 1)h)$.

Proof: We construct a sequence of trees D_i, $i \geq -1$, using the edge–centroid decomposition method. Each tree D_i has n_i tasks and height $\leq h$. Define D_0 to be T. For $i \geq 0$, if D_i has more than $\tau + \tau/c$ tasks, construct a tree D_{i+1} from D_i as follows. Decompose D_i into $\leq \lceil cdn_i/\tau \rceil$ subtrees $T_1, T_2, \ldots, T_j, \ldots$, each with $\leq \lceil \tau/c \rceil$ tasks. Tree D_{i+1} is the subtree of D_i induced by the tasks in D_i lying on a path from the root of a tree T_j to the root of D_i. Clearly, D_{i+1} has $n_{i+1} \leq \lceil cdn_i/\tau \rceil h$ tasks and height $\leq h$. Moreover, since $c < \tau/((d+1)h)$, we can show, by induction on i, that

$$n_i \leq (cdh/\tau)^i n + h \sum_{j=0}^{i-1} (cdh/\tau)^j \leq n(cdh/\tau)^i + h/(1 - cdh/\tau) \leq n(cdh/\tau)^i + \tau/c, \quad (28)$$

for each $i = 0, 1, 2, \ldots$.

Let k be the smallest positive integer such that $n_k \leq \tau + \tau/c$. Such a k exists as long as $cdh/\tau < 1$. From (28) it follows that

$$k \leq \frac{\log(n/\tau)}{\log(\tau/(cdh))} + 1. \quad (29)$$

Using the sequence $D_0, D_1, \ldots, D_k$ we describe a schedule for T. Consider a tree D_i, $0 \leq i \leq k-1$. Consider an edge–centroid decomposition of D_i into $\leq \lceil cdn_i/\tau \rceil$ subtrees $T_1, T_2, \ldots, T_j, \ldots$, each with $\leq \lceil \tau/c \rceil$ tasks. For each subtree T_j, we schedule all the tasks in $T_j - D_{i+1}$ on the same processor. All the tasks in $T_j - D_{i+1}$ can be executed in $\leq \lceil \tau/c \rceil$ time, and these tasks do not depend on tasks in D_i that belong to a different subtree. However, to ensure that the number of processors used to execute $D_i - D_{i+1}$ is $\leq \lceil 2cn_i/\tau \rceil$ we pack subtrees to processors so that each processor is assigned between $\lceil \tau/(2c) \rceil$ and $\lceil \tau/c \rceil$ tasks. Unfortunately, because tasks in D_{i+1} depend on tasks in $D_i - D_{i+1}$, we need to communicate the values of these tasks in $D_i - D_{i+1}$ to processors that need those values when executing tasks in D_{i+1}. 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 in $\lceil \tau/c \rceil + \tau$ time using $\leq \lceil 2cn_i/\tau \rceil$ processors. We assign all the tasks in D_k to one processor. Since D_k has $\leq \tau + \tau/c$ tasks, all its tasks can be executed in time $\tau + \tau/c$. (Note that processors are "re–used".)

Consequently, we can schedule T in $(\lceil \tau/c \rceil + \tau)(k+1)$ time using $\leq \lceil 2cn/\tau \rceil$ processors. The lemma follows from (29). ■

The previous upper bound in [9] for T_{PY} was the one given in (27) with $c = 1$ and $d = 2$. If $3h < \tau < n$ then, by choosing c in the range $[1, \tau/3h)$ so that the right hand side of (27) is minimized, we obtain a smaller upper bound on T_{PY}. Using the symbolic algebra package Maple [4] version V, release 2, running on a Sun SPARC-10 workstation of the Computer

[4] Copyright by the University of Waterloo.

Science Department at the University of Maryland Baltimore County, we found that the right hand side of (27) is minimized when $c = W(\tau/(edh))$, provided that $\tau \geq e^2 dh$, and where W is the function in [4]. This function satisfies $W(x)e^{W(x)} = x$. For example, if $\tau = 300e^2 \ln(n)$, $h = 10\ln(n)$, and $c = W(15e) \approx 2.711$ then for large enough n the upper bound is improved by a factor of ≈ 0.8684.

Appendix A

Kalpakis and Yesha [10] develop yet another way to decompose a tree into subtrees. The resulting decomposition is called *path–centroid decomposition*. Let T be a binary degree tree dag with n nodes and height h. Given a positive integer $\beta \leq n$, they 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.

They also 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.

They 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.

Decompose each subtree R_i that fails to satisfy the second property, into subtrees so that properties 1 and 2 are both 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 6 *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.

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] F. N. Fritsch and R. E. Shafer and W. P. Crowley. Solution of the Transcendental Equation $we^w = \chi[C5]$. *Communications of the ACM*, 16(2), pp. 123–124, 1973.

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

[6] 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.

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

[8] 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.

[9] K. Kalpakis and Y. Yesha. On the Power of the Linear Array Architecture for Performing Tree-Structured Computations. *Journal of Computer and System Sciences*, Vol. 50, No. 1, pp. 1–10, February 1995.

[10] K. Kalpakis and Y. Yesha. Scheduling Tree dags on Parallel Architectures. To appear in *Algorithmica*. Preliminary version available as UMIACS-TR-92-110.1 CS-TR-2974.1, University of Maryland at College Park, Institute for Advanced Computer Studies, October 1992.

[11] K. Kalpakis and Y. Yesha. Optimal within a Constant Schedules for Forest Dags on Parallel Architectures. In *Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems*, pp. 158–163, Princeton, New Jersey, October 1993.

[12] 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.

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

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

[15] 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.

Contents