Journal of Combinatorial Optimization 1, 129–149 (1997)

Scheduling Problems in a Practical Allocation Model*

LISA HOLLERMANN† hollerla@cda.mrs.umn.edu
IBM Corporation, Rochester, MN, USA

TSAN-SHENG HSU‡ tshsu@iis.sinica.edu.tw
Institute of Information Science, Academia Sinica, Nankang 11529, Taipei, Taiwan, ROC

DIAN RAE LOPEZ§ lopezdr@cda.mrs.umn.edu
Division of Science and Mathematics, University of Minnesota, Morris, Morris, Minnesota 56267, USA

KEITH VERTANEN¶ e96keive@und.ida.liu.se
Linkvping Universitet, Alsg. 15C:22, S-58435 Linkvping, Sweden

Received ; Revised

Abstract. A parallel computational model is defined which addresses I/O contention, latency, and pipe-lined message passing between tasks allocated to different processors. The model can be used for parallel task-allocation on either a network of workstations or on a multi-stage inter-connected parallel machine. To study performance bounds more closely, basic properties are developed for when the precedence constraints form a directed tree. It is shown that the problem of optimally scheduling a directed one-level precedence tree on an unlimited number of identical processors in this model is NP-hard. The problem of scheduling a directed two-level precedence tree is also shown to be NP-hard even when the system latency is zero.

An approximation algorithm is then presented for scheduling directed one-level task trees on an unlimited number of processors with an approximation ratio of 3. Simulation results show that this algorithm is, in fact, much faster than its worst-case performance bound. Better simulation results are obtained by improving our approximation algorithm using heusistics. Restricting the problem to the case of equal task execution times, a linear-time algorithm is presented to find an optimal schedule.

1. Introduction

Models for scheduling tasks on a parallel MIMD architecture have usually included a communication cost associated with the sending of data between tasks which are located on

*An extended abstract of part of this paper appeared in the Proceedings of the Seventh International Symposium on Algorithms and Computation, Osaka, Japan, 1996.

†Research supported by MAP Undergraduate Research Award from University of Minnesota, Morris. Lisa is currently an intern at IBM.

‡Research supported in part by NSC Grants 85-2213-E-001-003 and 86-2213-E-001-012.

§Research supported in part by Academia Sinica, Taipei, the University of Minnesota, Morris and the University of Minnesota China Center, Minneapolis.

¶Research supported by UROP grant from University of Minnesota, Morris. Keith is currently on study abroad at the University of Linkvping.

different processors. Early work on this problem used graph theoretic techniques such as network flow and/or enumeration techniques (El-Dissouki and Huen, 1980; Ma et al., 1982; Stone, 1977). Later work concentrated on approximation algorithms (Anger et al., 1990; Hwang et al., 1989; Lo, 1988; Papadimitriou and Yannakakis, 1990). Research then evolved to more restricted models which allowed an infinite number of processors in the system. Polynomial algorithms were found for the cases where the precedence constraints form a tree under certain constraints (Cheng and Sin, 1990; Chrétienne, 1989, 1992; Colin and Chrétienne, 1991, Lopez, 1992). A good review of models and algorithms developed for this problem can be found in (Cheng and Sin, 1990; Chrétienne et al., 1995; Lo, 1983; Norman and Thanisch, 1993). Most of this work was very theoretical in nature, i.e., the models were too simplistic for practical application to real machines. More recently, Valiant's BSP Model (Valiant, 1990a, 1990b) provided a general framework with which to study more practical algorithms in an asynchronous distributed memory parallel architecture. The LogP model (Culler et al., 1993) and the QRQW model (Gibbons et al., 1994) attempted to further bridge the theoretical and practical models.

This paper uses a practical and realistic model based on Valiant's asynchronous distributed memory architecture while taking into consideration the read/write contention of the QRQW model, the latency/overhead time of the LogP model, and the pipe-lined message sending cost which is proportional to the message size. The model can be used for a loosely-coupled parallel architecture where communication times are small but still significant. It is also general enough to represent a communication network of computers or workstations each with its own memory and microprocessor. The growth of such networks mandate more study into the efficient use of their parallel computing power. Unlike the LogP and QRQW models in which specific algorithms are designed to match the model, our model is general enough to be used for any algorithm which can be represented as a set of tasks which communicate with each other and whose execution and communication costs are known or can be estimated. An example where such an algorithm would be helpful is a network of computers using PVM parallel software (Geist et al., 1993). In today's environment, PVM program tasks are either scheduled by the programmer or, more often, they are arbitrarily allocated to processors(also called processing elements or PE's). The work of this paper is designed to allow the compiler and/or operating system to perform such tasks.

Previously, on a simpler and more theoretical model where message sending time is the only cost for communication, it has been shown that scheduling a two-level directed precedence tree (Chrétienne, 1994) and that scheduling a general directed precedence intree with task lengths (Lenstra et al., 1996) are both NP-hard. These important results show that we must either put constraints on the task set or develop approximation algorithms with good performance bounds. By constraining our model so that the set of tasks form only a one-level directed precedence tree and by allowing for communication costs for both the sending and receiving of messages, we prove that task allocation on even this more practical model is still NP-hard. We then proceed to develop an approximation algorithm for this case and to look at an even simpler case which does lend itself to a polynomial solution.

The results are summarized in Table 1.

Table 1. Summary of NP-hardness proofs and algorithms presented in this paper.

		Arbitrary task execution times	Equal task execution times
Directed one − level task trees w/n + 1 tasks	Approx. ratio	3 (NP-hard)	Optimal
	Running time	$O(n)$	$O(n)$ + sorting

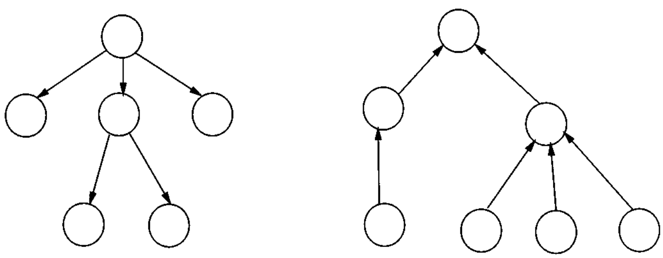
The paper is organized as follows. Section 2 defines the communication model used and develops some basic properties of the model. Section 3 proves NP-hardness results by a series of reductions from the knapsack problem. Section 4 considers algorithms for the case when the precedence constraints form a directed one-level tree. We give a 3-OPT approximation algorithm and an optimal liner-time algorithm for the special case of equal execution times of all tasks other than the root task in the system. Section 5 shows that the algorithms perform very close to optimal most of the time under simulated conditions. Conclusions follow.

2. The communication model

Let $J = \{t_0, t_1, \ldots, t_n\}$ be a set of tasks whose precedence constraints form a directed graph PC(J). In a precedence graph for a set of tasks, the weight on a directed edge $(u \rightarrow v)$ which points from u to v represents the communication time needed for u to send data to v if u and v are allocated on different processors. The weight on each node represents its execution time. In this model, we consider the case when all processors in the system are identical. Thus a task has the same execution time on any processor(PE) in the system.

A directed graph G is a *directed out-tree* if there is a vertex u in G such that there is exactly one directed path from u to any other vertex. The vertex u is the *root*. Each vertex in G having no outgoing edge is a *leaf*. A directed out-tree is a *k-level tree* if the length of the path from the root to each leaf is k. By reversing the direction of all edges in a directed out-tree, we obtain a *directed in-tree*. A *directed tree* is either a directed out-tree or a directed in-tree. An example is illustrated in figure 1.

Let $e(t_i) = e_i$ and $c(t_i) = c_i$ be the execution and communication time of the task t_i, respectively. For convenience, we define the *difference* of task t_i to be $d_i = e_i - c_i$. We



(1) (2)

Figure 1. In (1), a directed out-tree is illustrated. In (2), a directed in-tree is illustrated.

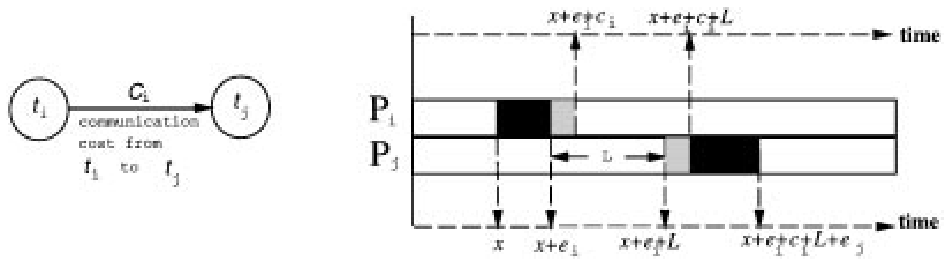


Figure 2. Task t_i is allocated to processor P_i and t_j is allocated to P_j. During time $x - (x + e_i)$, t_i is executed on P_i. The sending time from t_i to t_j is c_i. L, the system latency, is the units of time from when P_i starts to send data until P_j starts to receive the data from P_i, and the receiving time is c_i.

schedule J on uniform processors $P_0, P_1, \ldots, P_r$ with a system I/O latency, L. Note that $r \leq n$. In this model, task t_i takes e_i time to finish its computation and after its completion (might not be immediately) transmits data to the processor on which task t_j is allocated if there is a precedence relation from t_i to t_j. Task t_j cannot start executing unless it has received all data from t_i. We assume that the communication time is zero between two tasks allocated to the same processor. If t_i and t_j are allocated to different processors, then the sending time for t_i is c_j and the receiving time for t_j is also c_j. All data streams are transmitted in a pipelined fashion, i.e., after t_i starts sending, all data arrive at t_j in $c_j + L$ units of time. If a task needs to send or receive two data elements at the same time, the two I/O operations must take place in sequence. An example of a timing diagram for executing tasks in this model is shown in figure 2.

Realization of a scheduling. A scheduling, S, for J is an assignment of tasks to processors. A *legal realization* for S is the assignment of starting times for all tasks allocated to each processor such that it satisfies the precedence constraints and the I/O latency requirement. Given a realization, let $s(t_i)$ and $f(t_i)$ be, respectively, the start and finish execution times for t_i on the processor to which it is allocated. Let $s(c_i)$ and $f(c_i)$ be the start and finish times to send data to the processor t_i is located on. The *makespan* of a processor P_i for a realization is the time at which the processor P_i finishes all tasks allocated to it. The makespan of a legal realization is the largest makespan among all processors. A legal realization with the smallest makespan is a *best* realization. The makespan of a scheduling S is the makespan of its best realization and is denoted as $M(S)$. An *optimal scheduling J* is a scheduling with the smallest possible makespan. For convenience, we assume that t_0 is allocated to P_0. We now state a property which can be easily verified.

Lemma 2.1. *Let $J = \{t_0, t_1, \ldots, t_n\}$ be a set of tasks whose* PC(J) *forms a directed one-level tree with the root t_0. When scheduling J on an unlimited number of identical processors, there is an optimal scheduling where every processor, except the one on which t_0 is located, is allocated no more than one task.*

We next state a lemma with regard to properties of a best realization.

Lemma 2.2. *Let $J = \{t_0, t_1, \ldots, t_n\}$ be a set of $n+1$ tasks whose* PC(J) *forms a directed one-level out-tree with the root t_0 and whose execution times of tasks other than t_0 satisfy the condition $e_i \geq e_{i+1}$, $1 \leq i < n$. Given a scheduling for J, let $t_0, t_{v_1}, \ldots, t_{v_{n-w}}$ be tasks allocated to P_0 and let $t_{u_1}, t_{u_2}, \ldots, t_{u_w}$ be tasks not allocated to P_0. There exists a best realization for the given scheduling with the following properties*: (1) $s(t_0) = 0$; (2) $u_i < u_{i+1}$, $1 \leq i < w$; (3) $s(c_{u_i}) = e_0 + \sum_{j=1}^{i-1} c_{u_j}$, *for all* $1 \leq i \leq w$; (4) $s(t_{v_i}) = e_0 + \sum_{j=1}^{w} c_{u_j} + \sum_{j=w+1}^{i-1} e_{v_j}$, *for all* $1 \leq i \leq n - w$; (5) t_{u_i} *is allocated on* P_i *with* $s(t_{u_i}) = s(c_{u_i}) + L + c_{u_i}$, $1 \leq i \leq w$.

Proof: It is obviously true that any best realization executes t_0 on P_0 as soon as possible. After finishing the computation of t_0, executing other tasks allocated on P_0 before doing communication for t_0 does not decrease the makespan. Thus we may assume that all optimal realization makespans could execute t_{v_i} on P_0 only after t_0 sends all of its data to other processors. Let $f_i(R)$ be the finish time for t_i in a realization R. Let R be an optimal realization with some $e_{u_i} < e_{u_{i+1}}$. Let R' be the revised realization by swapping the order of sending data for t_{u_i} and $t_{u_{i+1}}$. The finish times for processors other than P_i and P_{i+1} are the same in R and R' and

$$\begin{aligned}
f_{u_i}(R) &= f(c_{u_{i-1}}) + c_{u_i} + L + e_{u_i};\\
f_{u_{i+1}}(R) &= f(c_{u_{i-1}}) + c_{u_i} + c_{u_{i+1}} + L + e_{u_{i+1}};\\
f_{u_i}(R') &= f(c_{u_{i-1}}) + c_{u_{i+1}} + L + e_{u_{i+1}};\\
f_{u_{i+1}}(R') &= f(c_{u_{i-1}}) + c_{u_i} + c_{u_{i+1}} + L + e_{u_i}.
\end{aligned}$$

Since $e_{u_i} < e_{u_{i+1}}$, thus $f_{u_{i+1}}(R) > f_{u_{i+1}}(R')$. It is always true that $f_{u_{i+1}}(R) \geq f_{u_i}(R')$. Thus the makespan for R' is no worse than the makespan for R. By using this lemma, we can find an optimal realization with the property that $e_{u_i} \geq e_{u_{i+1}}$, for all $1 \leq i < w$. □

An example for a best realization of a scheduling as described in Lemma 2.2 is illustrated in figure 3.

The symmetric property. In the following lemma, let $r(G)$ be the resulting graph obtained from a directed graph G by reversing the direction of each edge in G. The weights on nodes and edges remain the same. Note that if G is a directed tree, then $r(G)$ is also a directed tree.

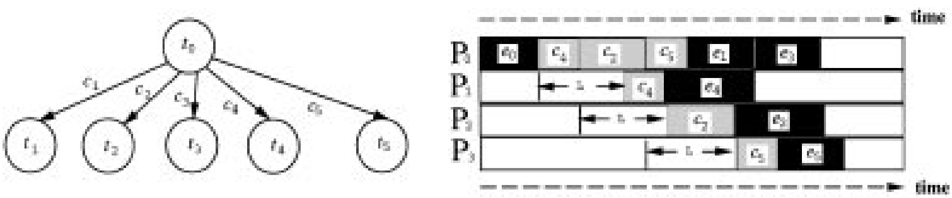


Figure 3. The form of a best realization for the precedence graph (shown above) when tasks t_0, t_1, and t_3 are assigned to P_0 and the rest of the tasks are each assigned to another processor. Note that e_i is the execution time for task t_i. In this given set of task, $L = 4$, $c_2 = 3$, $c_4 = 2$, $c_5 = 2$, $e_1 = 3$, $e_2 = 4$, $e_3 = 3$, $e_4 = 4$, and $e_5 = 3$. Since $e_4 \geq e_2 \geq e_5$, this is a best realization *for the above task assignment* according to Lemma 2.2

.

Lemma 2.3. *Let J be a set of tasks whose* $\mathrm{PC}(J)$ *is a directed tree. Let J' be the same set of tasks except that* $\mathrm{PC}(J') = r(\mathrm{PC}(J))$. *If there is a scheduling for J whose makespan is M, then there exists a scheduling for J' whose makespan is also M.*

Proof: Let S be a scheduling for J with a realization whose makespan is M. Since J and J' have the same set of tasks, S is also a scheduling for J'. Let R be a realization for S on J with the makespan M. We construct the realization R' for S on J' whose makespan is also M. Let $f_R(t_i)$ be the finish time for task t_i in R and let $a_R(t_i)$ be the finish time for task t_i to receive its needed data in R if the task sending that data is allocated to a processor that is different from the processor that t_i is on. Then $s_{R'}(t_i) = M - f_R(t_i)$ is the starting execution time for task t_i in R' and $s_{R'}(c_i) = M - a_R(t_i)$ is the starting time for task t_i to transmit data in R'. The makespan of R' is also M. □

Intuitively, in the proof of Lemma 2.3, we "reverse" the time arrow in R to derive R'.

The positive difference property. Let $J = \{t_0, t_1, \ldots, t_n\}$ be a set of tasks whose $\mathrm{PC}(J)$ is a directed tree rooted at t_0. We will show that a task whose difference (i.e., the execution time minus the communication time) is non-positive can be allocated on a processor with its parent to have an optimal scheduling.

Lemma 2.4. *Let S be a scheduling for a set of tasks whose* $\mathrm{PC}(J)$ *is a directed one-level tree. By re-allocating all tasks with non-positive differences to P_0, the resulting scheduling has equal or better makespan than that of S.*

Proof: By Lemma 2.3, we may assume that $\mathrm{PC}(J)$ is a one-level directed out-tree. Once we prove this case, the case when $\mathrm{PC}(J)$ is a directed in-tree follows.

Let t_w be a task with a non-positive difference which is allocated to a processor other than P_0 in an optimal scheduling. The parent of t_w is t_0 and t_0 is allocated on P_0. By re-allocating t_w on P_0, the makespan for P_0 is increased by d_w. Since $d_w \leq 0$, the makespan on P_0 does not increase. On the other hand, the makespan for P_i, $i > 0$, is also not increased. Thus the new scheduling is also optimal. We can continue this process until all tasks with non-positive differences satisfy the property specified in the lemma. □

3. NP-hardness results

A communication model where the sending time and I/O latency are both zero is a *simplified model*. A model that does not assume this is a *regular model*. In this section, we prove that the optimal scheduling problem for a set of tasks J is NP-hard in the regular model even when $\mathrm{PC}(J)$ is a directed one-level tree. The proof of the NP-hardness result is done by reducing the well-known knapsack problem to it. The proof is rather involved. We will also show that the proof holds when extended to the simplified model where $\mathrm{PC}(J)$ is a two-level directed tree.

3.1. Problem formulation

Definition 3.1. Let $J = \{t_0, t_1, \ldots, t_n\}$ be a set of tasks whose PC(J) is a directed one-level out-tree rooted at t_0.

(i) The decision problem OPTS(J, e, c, L, M) is as follows: Given a positive integer M, is there a scheduling for J whose makespan is less than or equal to M in a communication model with I/O latency L?
(ii) The decision problem K-OPTS(k, J, e, c, L, M) is as follows: Given positive integers k and M, is there a scheduling for J whose makespan is less than or equal to M using exactly k processors with exactly one task allocated on each of the $k-1$ processors in a communication model with I/O latency L?

Lemma 3.2. *The* OPTS(J, e, c, L, M) *problem is equivalent to the following problem: Does there exist an integer i that is at most n such that* K-OPTS(i, J, e, c, L, M) *has a "yes" answer?*

Lemma 3.3. *The* K-OPTS(k, J, e, c, L, M) *problem can also be formulated as follows: Is there an assignment of values to the set of binary variables $\{x_1, \ldots, x_n\}$ such that the following are satisfied?*

$$\sum_{i=1}^{n} x_i \cdot d_i \geq \sum_{i=1}^{n} e_i + e_0 - M, \tag{1}$$

and

$$\sum_{j \leq i} x_j \cdot c_j + (L + e_i) \cdot x_i \leq M - e_0, \quad \textit{for all } 1 \leq i \leq n \tag{2}$$

$$\sum_{i=1}^{n} x_i = k - 1 \tag{3}$$

Proof: If t_i is allocated on P_0, then $x_i = 0$. Otherwise, $x_i = 1$. The overall finish time on P_0 is $e_0 + \sum_{i=1}^{n}(1 - x_i) \cdot e_i + \sum_{i=1}^{n} x_i \cdot c_i$. This value must be less than or equal to M. This gives the first equation. The overall finish time on P_i, $0 < i \leq w$ is $e_0 + \sum_{j \leq i} x_j \cdot c_j + (L + e_i) \cdot x_i$. This value must be less than or equal to M. This gives the second equation. The third equation is trivial. □

3.2. OPTS(J, e, c, L, M) is NP-hard

We will prove that OPTS(J, e, c, L, M) is NP-hard even when PC(J) is a directed one-level out-tree by a reduction from the knapsack problem.

We first prove that a particular instance of the knapsack problem is NP-hard.

Definition 3.4.

(i) The decision version of the knapsack problem KNP(m, s, v, B, V) is as follows: Let M be a list of m elements where the ith element has positive size s_i and positive value v_i, and $s_i + v_i \geq s_{i+1} + v_{i+1}$, $1 \leq i < m$. Given two positive integers, B and V, is there a subset of elements $S \subseteq M$ such that $\sum_{i \in S} s_i \leq B$ and $\sum_{i \in S} v_i \geq V$?

(ii) The decision version of the knapsack problem with the cardinality constraint K-KNP(k, m, s, v, B, V) is as follows: Let M be a list of m elements where the ith element has positive size s_i and positive value v_i, and $s_i + v_i \geq s_{i+1} + v_{i+1}$, $1 \leq i < m$. Given three positive integers k, B and V, is there a subset of exactly k elements $S \subseteq M$ such that $\sum_{i \in S} s_i \leq B$ and $\sum_{i \in S} v_i \geq V$?

It is well-known that KNP(m, s, v, B, V) is NP-hard (Garey and Johnson, 1979). This problem is easily solvable in polynomial time if the values v_i are all the same or the sizes s_i are all the same. We now give a lemma, which leads to a corollary stating that the knapsack problem with the cardinality constraint is also NP-hard.

Lemma 3.5. *Given positive values $s_1, \ldots, s_n$, $v_1, \ldots, v_n$, B, V, and an integer k, let $\mathcal{V} = \max\{V, \sum_{i=1}^m v_i\} + 1$, $s'_i = s_i + q$, $v'_i = v_i + \mathcal{V}$, $B' = B + k \cdot q$, and $V' = V + k \cdot \mathcal{V}$. Let m be an integer such that $m > k + 2$. Then there exists a positive value q such that*

(i) $v'_i + s'_i \geq v'_{i+1} + s'_{i+1}$, $1 \leq i < m$,
(ii) $2 \cdot (v'_m + s'_m) \geq \sum_{i=1}^m v'_i - V' + 1 + (v'_1 + s'_1)$ *and*
(iii) $\sum_{i=1}^{m-1} (s'_i + v'_i) \geq B' + V' + 1$.

Proof: Let $q = \max\{B + \mathcal{V} + 1, v'_1 + s_1 - 2 \cdot s_m - v'_m + h\}$, where $h = \sum_{i=1}^{m-1} v'_i - V' + 1$. It is easy to see that $q > 0$.

(i) Thus $v'_i + s'_i = v_i + \mathcal{V} + s_i + q \geq v_{i+1} + \mathcal{V} + s_{i+1} + q = v'_{i+1} + s'_{i+1}$.
(ii) Note that $q \geq v'_1 + s_1 - 2 \cdot s_m - v'_m + h$.

$$\begin{aligned} 2 \cdot (v'_m + s'_m) - (h + v'_1 + s'_1) &= 2 \cdot v'_m + 2 \cdot (s_m + q) - h - v'_1 - s_1 - q \\ &= 2 \cdot v'_m + 2 \cdot s_m + q - h - v'_1 - s_1 \\ &\geq v'_m + 2 \cdot s_m + (v'_1 + s_1 - 2 \cdot s_m - v'_m + h) - h - v'_1 - s_1 \\ &= 0 \end{aligned}$$

Thus $2 \cdot (v'_m + s'_m) \geq h + v'_1 + s'_1 = \sum_{i=1}^m v'_i - V' + 1 + (v'_1 + s'_1)$.

(iii) Because $k < m - 2$, $\sum_{i=1}^{m-1} s'_i > \sum_{i=1}^{m-1} s_i + (k+1) \cdot q$. Note that $B' = B + k \cdot q$. Hence $\sum_{i=1}^{m-1} s'_i - B' \geq \sum_{i=1}^{m-1} s_i + (k+1)q - (B + k \cdot q) \geq q - B$. Since by definition $q \geq B + \mathcal{V} + 1$, $\sum_{i=1}^{m-1} s'_i \geq B' + \mathcal{V} + 1$. This implies $\sum_{i=1}^{m-1} (s'_i + v'_i) \geq B' + V' + 1$. □

Corollary 3.6. *The knapsack problem with the cardinality constraint is NP-hard.*

Proof: We transform the knapsack problem KNP(m, s, v, B, V) into the following problem: Given

- $\mathcal{V} = \max\{V, \sum_{i=1}^m v_i\} + 1$;
- $V' = V + k \cdot \mathcal{V}$;

- $q = \max\{B + \mathcal{V} + 1, v_1' + s_1 - 2 \cdot s_m - v_m' + \sum_{i=1}^{m-1} v_i' - V' + 1\}$,
- $B' = B + k \cdot q$;
- $s_i' = s_i + q, 1 \leq i \leq m$;
- $v_i' = v_i + \mathcal{V}, 1 \leq i \leq m$,

does there exist an integer i that is at most m and K-KNP(i, m, s', v', B', V') has a "yes" answer? Lemma 3.5 shows that K-KNP(i, m, s', v', B', V') is a valid instance for the knapsack problem with the cardinality constraint. It is easy to see that these two problems are equivalent. Thus the knapsack problem with the cardinality constraint is also NP-hard. □

Given an instance of the knapsack problem KNP(m, s, v, B, V), we know that we can obtain an instance of the knapsack problem with the cardinality constraint K-KNP(i, m, s', v', B', V'), $i \leq m$, as specified in the proof of Corollary 3.6.

Given an instance of K-KNP(k, m, s', v', B', V') as specified in the proof of Corollary 3.6, we then construct the following instance of the optimal scheduling problem K-OPTS$(k + 2, J, e, c, L, M)$ with tasks $t_0, t_1, \ldots, t_n$ and whose PC(J) forms a directed one-level out-tree rooted at t_0. Let $\mathcal{E}_i = \sum_{j=1}^{i} e_j$, let $\mathcal{C}_i = \sum_{j=1}^{i} c_j$, and let $\mathcal{D}_i = \sum_{j=1}^{i} d_j$.

- $n = m + 1$;
- $L = \mathcal{E}_{n-2} - B' - V'$;
- $d_i = v_i' = v_i + \mathcal{V}, 1 \leq i \leq n - 1$;
- $c_i = s_i' = s_i + q, 1 \leq i \leq n - 1$;
- $c_n = \mathcal{C}_{n-1} - 1 - L - B'$;
- $d_n = \mathcal{D}_{n-1} + 1 - V'$;
- $e_n = \mathcal{E}_{n-1} - L - B' - V'$;
- $M = \mathcal{C}_n + d_n + e_0 - 1$;

Lemma 3.7. K-OPTS$(k+2, J, e, c, L, M)$ *is a valid instance of the scheduling problem.*

Proof: We need to verify that $d_n > 0$, $e_n \geq 0$, and $L \geq 0$.

(i) Note that $v_i \geq 0$, $1 \leq i \leq n - 1$, $n \geq k + 2$ and $\mathcal{V} \geq V$.

$$\begin{aligned} d_n &= \mathcal{D}_{n-1} + 1 - V' \\ &= \left(\sum_{i=1}^{n-1} v_i \right) + (n-1) \cdot \mathcal{V} + 1 - V + k \cdot \mathcal{V} \\ &> 0 \end{aligned}$$

(ii) Note that $e_i = s_i' + v_i'$. Thus $e_1 \geq e_2 \geq \cdots \geq e_{n-1}$. Since

$$\begin{aligned} e_n &= \mathcal{E}_{n-1} - L - B' - V' \\ &= \mathcal{E}_{n-1} - (\mathcal{E}_{n-2} - B' - V') - B' - V' \\ &= e_{n-1}, \end{aligned}$$

$e_n \le e_{n-1}$. By Lemma 3.5, $2 \cdot e_{n-1} \ge \mathcal{D}_{n-1} - V' + 1 + e_1$. Starting from this assumption, we verify that $c_n = \mathcal{C}_{n-1} - 1 - L - B' \ge e_1 - e_n$.

$$
\begin{aligned}
2 \cdot e_{n-1} \ge \mathcal{D}_{n-1} - V' + 1 + e_1 & \\
&\Leftrightarrow \mathcal{C}_{n-1} + \mathcal{E}_{n-1} \ge 2 \cdot \mathcal{E}_{n-1} - V' - 2 \cdot e_{n-1} + 1 + e_1 \\
&\Leftrightarrow \mathcal{C}_{n-1} + \mathcal{E}_{n-1} \ge 2 \cdot \mathcal{E}_{n-2} - V' + 1 + e_1 \\
&\Leftrightarrow \mathcal{C}_{n-1} - 1 - 2 \cdot B' - V' + \mathcal{E}_{n-1} \ge 2 \cdot \mathcal{E}_{n-2} - 2 \cdot B' - 2 \cdot V' - 1 + 1 + e_1 \\
&\Leftrightarrow \mathcal{C}_{n-1} - 1 - 2 \cdot B' - V' + \mathcal{E}_{n-1} \ge 2 \cdot L + e_1 \\
&\Leftrightarrow \mathcal{C}_{n-1} - 1 - L - B' \ge e_1 - (\mathcal{E}_{n-1} - L - B' - V') \\
&\Leftrightarrow c_n \ge e_1 - e_n
\end{aligned}
$$

Thus $c_n \ge e_1 - e_{n-1} \ge 0$.

(iii) By Lemma 3.5, $\mathcal{E}_{n-2} \ge B' + V' + 1$. This implies $L \ge 0$. □

The following two lemmas shows that these two problems are equivalent.

Lemma 3.8. *If $x_n = 0$ in the solution vector for* K-OPTS$(k+2, J, e, c, L, M)$ *as formulated in Lemma* 3.3, *then we cannot answer "yes" to the above decision problem whose* PC(J) *is a directed one-level out-tree.*

Proof: Assume that that $x_n = 0$. Then $\sum_{i=1}^{n} d_i \cdot x_i = \sum_{i=1}^{n-1} d_i \cdot x_i \le \mathcal{D}_{n-1}$. Equation (1) in Lemma 3.3 gives $\sum_{i=1}^{n} d_i \cdot x_i \ge \mathcal{E}_n + e_0 - M$. Note that $M = \mathcal{C}_n + d_n + e_0 - 1$. Thus $\sum_{i=1}^{n} d_i \cdot x_i \ge \mathcal{D}_{n-1} + 1$. Hence it is impossible to have $x_n = 0$ if we want to have an "yes" answer. □

Lemma 3.8 states that in order for K-OPTS$(k+2, J, e, c, L, M)$ to have a "yes" answer, t_n must not be allocated on P_0.

A solution for a K-KNP(k, m, s', v', B', V') problem can be formulated as finding a vector $\langle x_1, \ldots, x_m \rangle$, such that $x_i = 1$ if the ith item is selected in the knapsack.

Lemma 3.9. *A solution vector $\langle \bar{x}_1, \ldots, \bar{x}_m \rangle$ for* K-KNP(k, m, s', v', B', V') *is equivalent to a solution vector $\langle \bar{x}_1, \ldots, \bar{x}_{n-1}, 1 \rangle$ as formulated in Lemma* 3.3 *for* K-OPTS$(k+2, J, e, c, L, M)$ *whose* PC(J) *is a directed one-level out-tree, if $k < m - 2$.*

Proof: Note that $m = n - 1$. By Lemma 3.7, K-OPTS$(k+2, J, e, c, L, M)$ is a valid instance for a scheduling problem.

We divide our proof into two parts.

Part (i): We first verify a solution vector $\langle \bar{x}_1, \ldots, \bar{x}_n \rangle$ for the scheduling problem K-OPTS $(k+2, J, e, c, L, M)$ gives a solution vector $\langle \bar{x}_1, \ldots, \bar{x}_m \rangle$ for K-KNP(k, m, s', v', B', V'). That is, given $\langle \bar{x}_1, \ldots, \bar{x}_n \rangle$ as formulated in Lemma 3.3, we need to verify that $\sum_{i=1}^{m} v'_i \cdot \bar{x}_i \ge V'$ and $\sum_{i=1}^{m} s'_i \cdot \bar{x}_i \le B'$.

By Eq. (1) in Lemma 3.3 and the fact that $\bar{x}_n = 1$ (Lemma 3.8), we know that

$$\begin{aligned}
\sum_{i=1}^{m} v_i' \cdot \bar{x}_i &= \sum_{i=1}^{n-1} d_i \cdot \bar{x}_i \\
&\geq e_0 + \mathcal{E}_n - M - d_n \\
&= e_0 + \mathcal{E}_n - (\mathcal{C}_n + d_n + e_0 - 1) - d_n \\
&= \mathcal{D}_n - 2 \cdot d_n + 1 \\
&= \mathcal{D}_{n-1} - d_n + 1 \\
&= \mathcal{D}_{n-1} + V' - V' - d_n + 1 \\
&= V' + (\mathcal{D}_{n-1} + 1 - V') - d_n \\
&= V'
\end{aligned}$$

From Eq. (2) in Lemma 3.3 (by setting $i = n$) and the fact that $\bar{x}_n = 1$ (Lemma 3.8),

$$\begin{aligned}
\sum_{i=1}^{m} s_i' \cdot \bar{x}_i &= \sum_{i=1}^{n-1} c_i \cdot \bar{x}_i \\
&\leq M - e_0 - L - e_n - c_n \\
&= (\mathcal{C}_n + (e_n - c_n) + e_0 - 1) - e_0 - L - e_n - c_n \\
&= \mathcal{C}_{n-1} - 1 - L - c_n \\
&= \mathcal{C}_{n-1} - 1 - L - (\mathcal{C}_{n-1} - 1 - L - B') \\
&= B'
\end{aligned}$$

Part (ii): We now verify that a solution vector $\langle \bar{x}_1, \ldots, \bar{x}_m \rangle$ for K-KNP(k, m, s', v', B', V') gives a solution vector $\langle \bar{x}_1, \ldots, \bar{x}_n \rangle$ for the scheduling problem K-OPTS$(k+2, J, e, c, L, M)$. That is, given the fact that $\sum_{i=1}^{m} v_i' \cdot \bar{x}_i \geq V'$ and $\sum_{i=1}^{m} s_i' \cdot \bar{x}_i \leq B'$, we must derive the three equations in Lemma 3.3.

$$\begin{aligned}
\sum_{i=1}^{n} d_i \cdot \bar{x}_i &= \sum_{i=1}^{m} v_i' \cdot \bar{x}_i + d_n \\
&\geq V' + d_n \\
&= V' + \mathcal{D}_{n-1} + 1 - V' \\
&= \mathcal{D}_{n-1} + 1 \\
&= \mathcal{D}_n + 1 - d_n \\
&= \mathcal{E}_n + e_0 + 1 - d_n - \mathcal{C}_n - e_0 \\
&= \mathcal{E}_n + e_0 - (\mathcal{C}_n + d_n + e_0 - 1) \\
&= \mathcal{E}_n + e_0 - M
\end{aligned}$$

$$\begin{aligned}
\sum_{i=1}^{n} c_i \cdot \bar{x}_i + (L + e_n) \cdot \bar{x}_n &= \sum_{i=1}^{m} s_i' \cdot \bar{x}_i + c_n + L + e_n \\
&\leq B' + c_n + L + e_n
\end{aligned}$$

$$
\begin{aligned}
&= B' + \mathcal{C}_{n-1} - 1 - L - B' + L + e_n \\
&= \mathcal{C}_{n-1} + e_n - 1 + c_n - c_n \\
&= \mathcal{C}_n + d_n - 1 + e_0 - e_0 \\
&= M - e_0
\end{aligned}
$$

In the following equations, i is any integer less than n. Recall that $e_1 \geq e_i$ and $c_n = \mathcal{C}_{n-1} - 1 - L - B' \geq e_1 - e_n$. Thus $c_n + e_n \geq e_1$.

$$
\begin{aligned}
\sum_{j \leq i} c_j \cdot \bar{x}_j + (L + e_i) \cdot \bar{x}_i &\leq \sum_{j=1}^{n-1} c_j \cdot \bar{x}_j + c_n + (L + e_n) \\
&= \sum_{j=1}^{n} c_j \cdot \bar{x}_j + (L + e_n) \cdot \bar{x}_n \\
&\leq M - e_0
\end{aligned}
$$

Since $\bar{x}_n = 1$ and $\sum_{i=1}^{m} \bar{x}_i = k$, thus $\sum_{i=1}^{n} = k + 1$. □

Theorem 3.10. *The decision version of the* OPTS(J, e, c, L, M) *problem is NP-hard even when* PC(J) *is a directed one-level out-tree.*

Proof: By Corollary 3.6 and Lemma 3.9. □

Corollary 3.11. *The decision version of the* OPTS(J, e, c, L, M) *problem is NP-hard even when* PC(J) *is a directed one-level in-tree.*

Proof: This corollary follows from Lemma 2.3 and Theorem 3.10. □

3.3. Other NP-hard instances

Definition 3.12. A directed graph $G = (V, E)$ is a *HARPOON graph* of size n if the set of vertices $V = \{w, A_1, \ldots, A_n, B_1, \ldots, B_n\}$ and the set of edges $E = \{(w \to A_i) \mid 1 \leq i \leq n\} \cup \{(A_i \to B_i) \mid 1 \leq i \leq n\}$, where $(u \to v)$ denotes a directed edge pointed from vertex u to vertex v. The vertex w is the root of G. Vertices in $\{A_1, \ldots, A_n\}$ are *leading vertices* and vertices in $\{B_1, \ldots, B_n\}$ are *tailing vertices*.

Note that G is a directed two-level out-tree in the above definition. An example for a directed two-level out-tree in illustrated in in figure 4.

For discussion here, let $J' = \{w, A_1, \ldots, A_n, B_1, \ldots, B_n\}$ be a set of tasks whose PC(J') is a HARPOON graph with the root w, leading vertices $\{A_1, \ldots, A_n\}$, and tailing vertices $\{B_1, \ldots, B_n\}$. The root task is w. The communication time from w to any leading task is large enough such that for any optimal scheduling all leading tasks are allocated on the processor where the root task w is allocated.

We use the following notation for tasks in J':

- $g_1(w)$ is the execution time of task w;
- $g_1(A_i)$ is the execution time of task A_i;

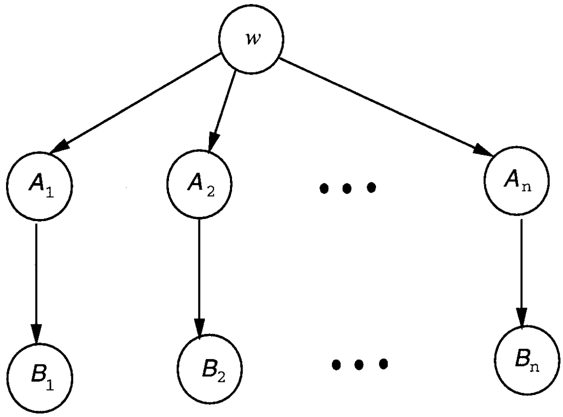


Figure 4. A directed two-level out-tree (called a HARPOON graph in Chrétienne (1994).

- $g_1(B_i)$ is the execution time of task B_i;
- $g_2(A_i)$ is the communication time needed to send data from task w to task A_i if w and A_i are allocated on different processors;
- $g_2(B_i)$ is the communication time needed to send data from task A_i to task B_i if A_i and B_i are allocated on different processors.

Lemma 3.13. *The decision version of the optimal scheduling problem* $\mathrm{OPTS}(J', g_1, g_2, 0, M)$ *is NP-hard in the simplified model with* $\mathrm{PC}(J')$ *being a two-level directed out-tree, where g_1 is the function to map a task to its execution time and g_2 is the function to map a task to the amount of communication time needed to receive its data.*

Proof: Let $J = \{t_0, t_1, \ldots, t_n\}$ be a set of tasks to be scheduled in the regular model and whose $\mathrm{PC}(J)$ is a directed one-level out-tree. We will prove that if $\mathrm{OPTS}(J', g_1, g_2, 0, M)$ is solvable in polynomial time in the simplified model, then $\mathrm{OPTS}(J, e, c, L, M)$ is also solvable in polynomial time.

Given $\mathrm{OPTS}(J, e, c, L, M)$ in the regular model, we construct $\mathrm{OPTS}(J', g_1, g_2, 0, M)$ in the simplified model with the properties that $g_1(w) = e_0$, $g_1(A_i) = c_i$, $g_1(B_i) = e_i - c_i$, $g_2(A_i) = c_i$, and $g_2(B_i) = c_i + L$. A scheduling for $\mathrm{OPTS}(J', g_1, g_2, 0, M)$ in the simplified model naturally corresponds to a scheduling for $\mathrm{OPTS}(J, e, c, L, M)$ in the regular model.

By Theorem 3.10, $\mathrm{OPTS}(J, e, c, L, M)$ is NP-hard in the regular model. Hence $\mathrm{OPTS}(J', g_1, g_2, 0, M)$ is also NP-hard in the simplified model. □

Note that a result that is similar to the one stated in Lemma 3.3 on a model without latency and I/O contention is first described in Chrétienne (1994) by a transformation from the knapsack problem that is as complex as the one stated in this paper. By using our result in Section 3.2, we can easily derive Lemma 3.13 on a model without latency, but enforcing I/O contention rules.

Corollary 3.14. *The decision version of the optimal scheduling problem* $\mathrm{OPTS}(J', g_1, g_2, 0, M)$ *is NP-hard in the simplified model with* $\mathrm{PC}(J')$ *being a two-level directed in-tree.*

Proof: This is a corollary of Lemmas 2.3 and 3.13. □

4. Algorithms for scheduling directed one-level task trees

Given an NP-hard problem, two approaches present themselves: 1) try to approximate the solution with a fast polynomial algorithm or 2) try to restrict the problem such that an optimal polynomial solution can be found. In this section, we take both approaches. Section 4.1 gives an approximation algorithm and Section 4.2 gives an optimal algorithm for a restricted case.

Consider the case of scheduling a directed one-level task tree using an unlimited number of processors. By Lemma 2.3, we need only consider task graphs that are directed one-level out-trees. Let $J = \{t_0, t_1, \ldots, t_n\}$ be a set of tasks whose $\mathrm{PC}(J)$ is a directed one-level out-tree rooted at t_0. Let e_i and c_i be the execution and communication time of task t_i, respectively. Let L be the system I/O latency. We schedule J on h identical processors which are denoted as $P_0, P_1, \ldots, P_{h-1}$.

4.1. *Scheduling with arbitrary task execution times*

We describe below an approximation algorithm for scheduling directed one-level task out-trees on an unlimited number of processors. This is an NP-hard scheduling problem by Theorem 3.10

We use the following notation: $E' = \sum_{t_i \ni e_i \le c_i} e_i$, and $C'' = \sum_{t_i \ni e_i > c_i} c_i$. Without loss of generality, assume that t_0 is allocated on processor P_0. We first give a lemma to help bound from below the value of $\mathrm{OPT}(J)$, the optimal makespan for J on an unlimited number of identical processors.

Lemma 4.1. (i) *An optimal scheduling for* J *is to schedule all tasks on* P_0 *if and only if for all tasks* t_i *with* $i > 0$ *and* $e_i > c_i$, $\sum_{i=1}^{n} e_i \le c_i + L + e_i$. (ii) *If scheduling all tasks on* P_0 *is not an optimal scheduling*, *then* $\mathrm{OPT}(J) > e_0 + L$. (iii) $\mathrm{OPT}(J) \ge e_0 + E' + C''$. (iv) $\mathrm{OPT}(J) \ge e_i$, $0 \le i \le n$.

Proof:

(i) The "only if" part of the proof is trivial since putting a task on another processor in this case only increases the makespan. We now prove the "if" part.

Let S be an optimal scheduling with all tasks allocated to P_0. Thus the makespan of S is $e_0 + \sum_{i=1}^{n} e_i$. Assume that there is a scheduling S' with at least one task t_w with $1 \le w \le n$ and $e_w > c_w$ such that $\sum_{i=1}^{n} e_i > c_w + L + e_w$. We know that $e_0 + \sum_{i=1}^{n} e_i - e_w + c_w < e_0 + \sum_{i=1}^{n} e_i$ since $e_w > c_w$ and that $e_0 + c_w + L + e_w < e_0 + \sum_{i=1}^{n} e_i$ by our assumption. This implies that $M(S') < M(S)$ which is a contradiction since S was an optimal scheduling. The conclusion follows.

(ii) If scheduling all tasks on P_0 is not an optimal scheduling, then we must at least schedule one task t_i, $i > 0$, on processor P_i. The makespan of P_i is at least $e_0 + c_i + L + e_i$. Thus this part of the lemma holds.

(iii) By Lemma 2.4, we know that scheduling tasks with $e_i > c_i$ on a processor other than P_0 does not improve the makespan. Thus all such tasks can be scheduled on P_0. The minimum makespan on P_0 for any scheduling is at least equal to $e_0 + E' + C''$.

(iv) This part is trivial. □

Using Lemma 4.1, our simple 3-OPT approximation algorithm to find a scheduling works as follows.

Algorithm A /* a scheduling on at most $n + 1$ processors. */
1. Check whether scheduling all tasks on P_0 is an optimal scheduling (Lemma 4.1).
2. Otherwise, allocate a task t_i, $i \neq 0$, with $e_i \geq c_i$ to P_i by itself, and the rest of the tasks to P_0;

Lemma 4.2. (i) *Algorithm A runs in $O(n)$ time.* (ii) *The makespan of the scheduling produced by Algorithm* A *is less than three times the optimal makespan.*

Proof: Part (i) is trivial. We prove part (ii). Note that if the condition in Step 1 holds, then Algorithm A finds an optimal scheduling by part (i) in Lemma 4.1. Thus we look at the case where the condition in Step 1 fails. Let S be the scheduling produced in Step 2. The makespan of P_0 in S is $e_0 + E' + C''$ which is at most $\mathrm{OPT}(J)$ by part (iii) in Lemma 4.1. The makespan of P_i, $i > 0$ and $e_i > c_i$, is less than or equal to $e_0 + C'' + L + e_i$. Therefore, we note that $e_0 + C''$ is no more than $\mathrm{OPT}(J)$ by part (iii) in Lemma 4.1, L is less than $\mathrm{OPT}(J)$ by part (ii) in Lemma 4.1, and e_i is also no more than $\mathrm{OPT}(J)$ by part (iv) in Lemma 4.1. Thus the makespan of any processor is less than $3 \cdot \mathrm{OPT}(J)$. □

4.2. Scheduling with equal task execution times

In this section, we consider the problem of finding an optimal scheduling for directed one-level task out-trees when the execution times of non-root tasks are restricted to be equal. We show an algorithm for finding an optimal scheduling using an unlimited number of processors.

Assume that all execution times are the same, i.e., $e_i = e_j = e$, $1 \leq i, j \leq n$. In this section, we assume without loss of generality that $c_i \leq c_{i+1}$, $1 \leq i < n$. Note that the difference d_i equals $e_i - c_i$. We also assume for now that $d_i > 0$, $1 \leq i \leq n$.

Lemma 4.3. *There is an integer p such that allocating $t_0, t_{p+1}, \ldots, t_n$ to processor P_0 and allocating tasks t_i, $1 \leq i \leq p$, to processors other than P_0 is an optimal scheduling for J.*

Proof: Let S be an optimal scheduling for J. By Lemma 2.2, a subset of tasks are allocated to P_0, while each of the remaining tasks is allocated to a processor by itself. Without loss of generality, assume that task t_i is allocated to P_i, if t_i is not allocated to P_0.

If S is not formed by allocating tasks $t_0, t_{k+1}, \ldots, t_n$ on processor P_0 and allocating tasks t_i, $1 \leq i \leq k$, to processors other than P_0, then there is a task t_i allocated on P_0 and another task t_j, $j > i$, which is not allocated on P_0. Let x be the smallest integer such that task t_x is allocated on P_0. Let y be the smallest integer that is greater than x and t_y is allocated on P_{a_y}, where $a_y \neq 0$. We construct another scheduling S' by taking S and applying the following task re-allocations: task t_x is re-allocated on processor P_{a_y} and task t_y is re-allocated on processor P_0. Let S' be this resulting scheduling. By Lemma 2.2, Processor P_0 first executes t_0. Since the execution times of all non-root tasks are equal, P_0 can send out data to tasks not allocating on P_0 in arbitrary order. Thus we may assume that our algorithm uses increasing order of task number for best realizations of S and S'. The makespan on P_0 in S' is not larger than the makespan on P_0 for S since $d_x > 0$ and $d_y \leq d_x$. The makespan of P_{a_y} in S' is less than the makespan of P_{a_y} in S, since $c_x + L + e_x \leq c_y + L + e_y$. Thus the makespan of S' is no greater than the makespan of S. We can continue to apply this process until the resulting schedule is of the form desired. □

Note that a similar proof for a simpler and more theoretical model where message sending time is the only cost for communication was given in Chrétienne (1994). An example for an optimal scheduling specified in Lemma 4.3 is illustrated in figure 5.

Algorithm E /* Scheduling on $n+1$ processors with $e_1 = \cdots = e_n = e$. */

1. **if** there is an r such that $r > 0$ and $L + e = (n - r) \cdot e$,
   then $k = r$; /* The makespan of P_r = Makespan of P_0. */
2. **else**
   (a) **if** $L + e > (n - 1) \cdot e$,
       then $r = 0$; /* Schedule all tasks on P_0. */
   (b) **else** find an integer r such that $L + e < (n - r) \cdot e$ and $L + e > (n - r - 1) \cdot e$;
       endif; /* Note that $(n - r - 1) = (n - (r + 1)$ */
   /* $e_0 + \sum_{i=1}^{r} c_i + (n - r) \cdot e$ is the schedule makespan of allocating $t_{r+1}, \ldots, t_n$ on P_0, which derives LHS of the inequality in 2c. */
   /* $e_0 + \sum_{i=1}^{r+1} c_i + L + e$ is the schedule makespan of allocating $t_{r+2}, \ldots, t_n$ on P_0, which derives RHS of the inequality in 2c. */
   (c) **if** $(n - r) \cdot e \leq c_{r+1} + L + e$,
       then $k = r$;
   (d) **else** $k = r + 1$;
       endif;
   endif;
3. Allocate tasks $t_0, t_{k+1}, \ldots, t_n$ on P_0; Allocate task t_i, $1 \leq i \leq k$, on P_i;

Theorem 4.4. *The optimal solution to the directed one-level precedence tree scheduling problem can be found in linear time when the execution times of all non-root tasks are equal and tasks are sorted according to their communication costs.*

Proof: The algorithm is shown in Algorithm E. Let S_k be the scheduling formed by allocating tasks $t_0, t_{k+1}, \ldots, t_n$ on P_0 and allocating t_i on P_i, $1 \leq i \leq k$. Let T_i be the

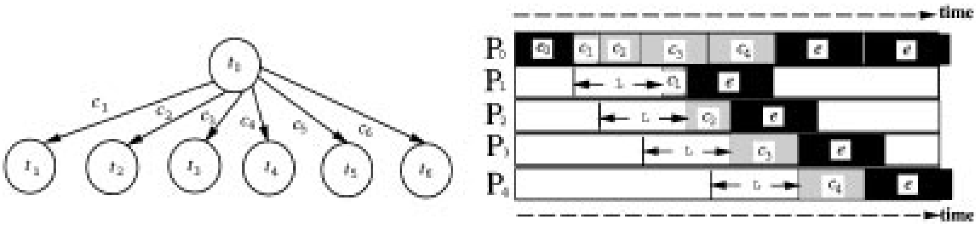


Figure 5. An optimal scheduling by allocating tasks t_0, t_5, and t_6 to processor P_0. In this given set of tasks, $c_1 = 1$, $c_2 = 2$, $c_3 = c_4 = c_5 = c_6 = 3$, $e_i = e = 4$, $1 \leq i \leq 6$, and $L = 4$. Note that $c_i \leq c_{i+1}$, $1 \leq i < 6$.

makespan of processor P_0 for S_i. Then T_i is a monotonically decreasing sequence, since $d_i > 0$, $1 \leq i \leq n$. Note that $T_0 > 0$.

By Lemma 2.2 and the fact that $e_i = e_j$ for all i and j, the makespan of processor P_i is greater than or equal to the makespan of P_j in an optimal scheduling if both i and j are not allocated on P_0 and $i > j$. Let W_i be the makespan among processors $P_1, \ldots, P_n$ in the scheduling S_i. Then W_i is a monotonically increasing sequence. Note that $W_0 = 0$.

$M_i = \max\{T_i, W_i\}$. Let $M^* = \min_{i=1}^{n} M_i$. Thus either $M^* = M_k$ where k is an integer and $T_k = W_k$ or, if $T_i \neq W_i$ for all i, then either $M^* = M_r$ or $M^* = M_{r-1}$ where r is an integer with $T_{r-1} > W_{r-1}$ and $T_r < W_r$. $\square$

If the input tasks are not previously sorted according to their communication times, then an optimal algorithm takes $O(n \cdot \log n)$ time.

5. Simulation results

We have provided a practical and realistic model with algorithms and worst-case performance bounds. We now attempt to provide some answers to questions such as: But how do these algorithms perform on the average? How often do they actually reach that worst-case scenario? Are they worthy of consideration for usage with existing computing resources?

Our experiments used randomly generated data for the communication (c_i) and execution (e_i) costs as well as the latency time, L. We specified the bounds of c_i and e_i to be uniformly distributed between 0.1 and 10.0 and the latency to be uniformly distributed between 0.2 and 8.0. We also tried widely varying latency bounds to simulate processors which were great distances from one another.

We first computed the optimal algorithm for a set of random data by using a brute force algorithm. Because of time constraints produced by computing the optimal algorithm, we only worked with sets of tasks less than twenty. After computing the optimal algorithm, we simulated our algorithm on the set of tasks and then compared the results. For each size of task set, we computed the optimal and approximation result for 5000 sets of random data. We then compared our algorithm result against the optimal result. We found that for

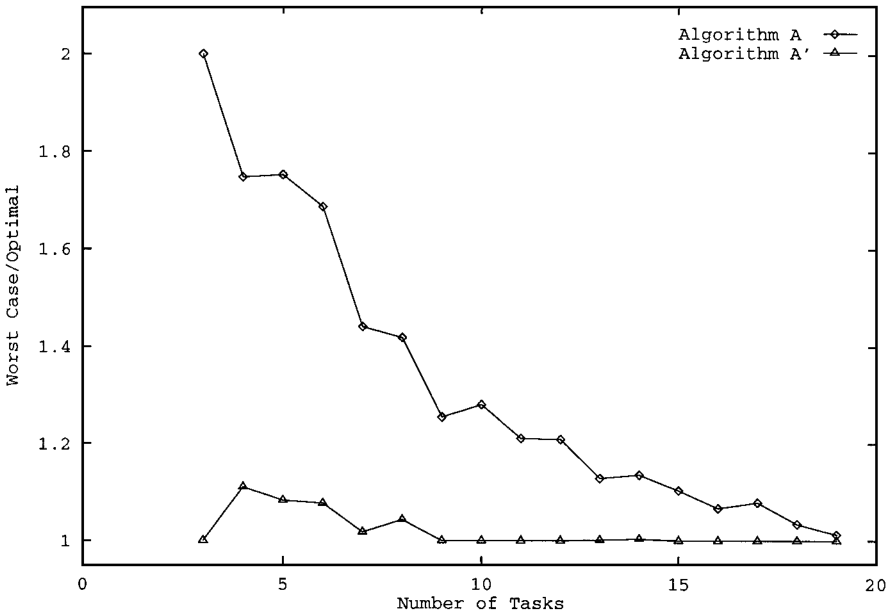


Figure 6. Worst case found for 5000 random simulations for each number of tasks in Algorithm A.

greater than ten tasks, Algorithm A always performed at less than 1.2 times the optimal (see figure 6).

Recall that our worst-case performance bound was three times the optimal for the case of using an unlimited number of processors. In practice, Algorithm A only found one set of tasks where the worst case was greater than 2.0 in the 17 sets of 5,000 random simulations each for numbers of tasks between three and nineteen. In fact, 90% of the approximation schedules were identical to the optimal schedule when using ten or more tasks in a set (see figure 7).

By varying the *L* parameter widely, we came up with only slightly larger bounds for small sets of tasks but they all followed the same pattern as did our specified parameter range results.

The average case using Algorithm A was even more gratifying, ranging between 1.0 and 1.08 times the optimal (see figure 8).

During our experiments for Algorithm A, we found that small task sets can provide uneven results. Placing only one task on the ‘wrong’ processor to give a non-optimal solution can make a large difference in the makespan when we are considering only a few tasks. For example, in figures 8 and 7, the performance curve for when the number of tasks is less than five does not seem to conform to the general trend of data that we obtained. In examining our schedules produced by Algorithm A, we discovered that it was often the case that only one task was out of place to produce a non-optimal schedule. Using this knowledge, we modified Algorithm A as follows.

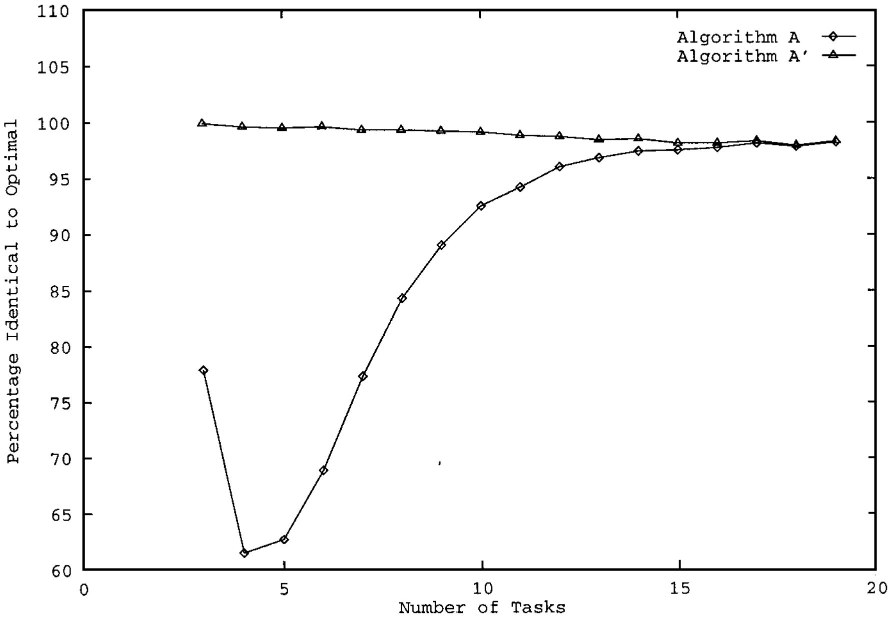


Figure 7. Percentage of Algorithm A and Algorithm A′ schedules identical to the optimal.

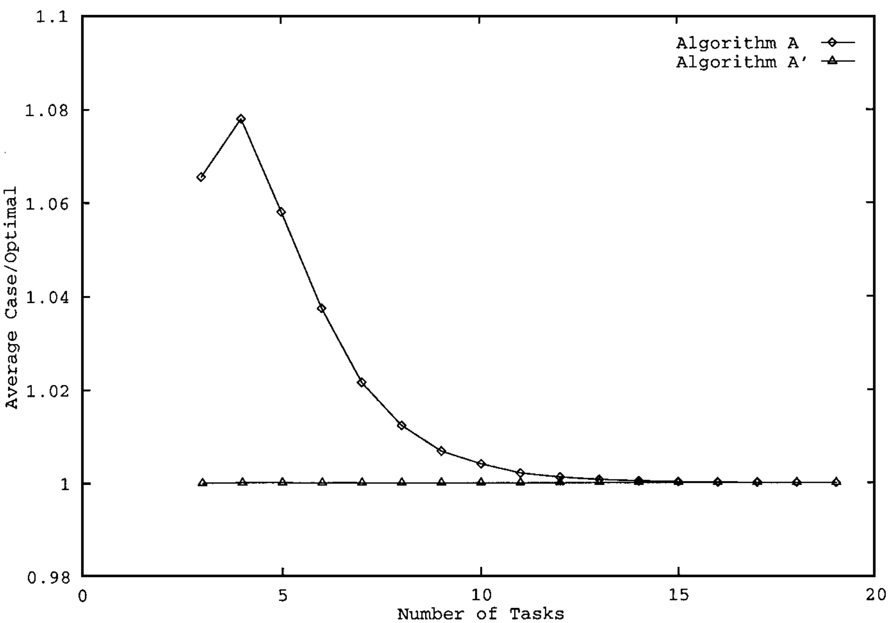


Figure 8. Average case for each 5000 trials for Algorithm A and Algorithm A′.

Algorithm A′ /∗ Modification of Algorithm A. ∗/
Add the following after Step 2 of Algorithm A:
3. Let the scheduling we have so far be R;
4. For each task t_i, $1 \leq i \leq n$, placed on P_i in R do:
   Modify R by placing t_i on P_0;
   Let the makespan of the resulting scheduling be M_i;
5. Compare makespans M_i, $1 \leq i \leq n$, with the makespan of R and use the schedule with the smallest makespan.

Algorithm A′ always gives a better solution than Algorithm A (see figure 6). Algorithm A′ also gives 'almost optimal' results even for small task sets (see figure 8). Note that adding this improvement to the algorithm does not change its linear time complexity. The percentage of identical schedules is, of course, also much better when using Algorithm A′ (see figure 7).

In summation, the average case simulation results are extremely close to optimal and even looking at worst cases and varying parameters, we obtain excellent results.

6. Concluding remarks

A practical and realistic model is presented for allocating tasks in a parallel distributed memory architecture. The model is designed to handle any algorithm for which task execution and communication costs can be known or estimated in advance. Our proof of NP-hardness when the precedence constraints form a directed one-level tree and the fundamental properties developed for this model together open the door for the design of good approximation algorithms both for scheduling an unbounded number of available processors and for scheduling a lesser fixed number of available processors in the system. We have demonstrated the design of an approximation algorithm and a polynomial time algorithm for one-level precedence trees. We have also demonstrated that the approximation algorithm performs very close to optimal under simulated conditions. This is a starting point for finding more tractable algorithms under less stringent conditions. Such work can eventually be used by a compiler to allocate the tasks of a general algorithm to execute in parallel efficiently.

References

F.D. Anger, J.J. Hwang, and Y.C. Chow, "Scheduling with sufficient loosely coupled processors," *Journal of Parallel and Distributed Comput.*, vol. 9, pp. 87–92, 1990.

T.C.E. Cheng and C.C.S. Sin, "A state-of-the-art review of parallel-machine scheduling research," *European J. Operational Research*, vol. 47, pp. 271–292, 1990.

P. Chrétienne, "A polynomial algorithm to optimally schedule tasks on a virtual distributed system under tree-like precedence constraints," *European J. Operational Research*, vol. 43, pp. 225–230, 1989.

P. Chrétienne, "Task scheduling with interprocessor communication delays," *European J. Operational Research*, vol. 57, pp. 348–354, 1992.

P. Chrétienne, "Tree scheduling with communication delays," *Discrete Applied Math.*, vol. 49, pp. 129–141, 1994.

P. Chrétienne, Jr., E.G. Coffman, J.K. Lenstra, and Z. Liu (Eds.), *Scheduling Theory and its Applications*, John Wiley & Sons Ltd., 1995.

J.Y. Colin and P. Chrétienne, "C.P.M. scheduling with small communication delays and task duplication," *Oper. Res.*, vol. 39, no. 3, pp. 680–684, 1991.

D. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, and T. von Eicken, "LogP: Towards a realistic model of parallel computation," in *Proc. 4th ACM SIGPLAN Symp. on Principles and Practices of Parallel Programming*, 1993, pp. 1–12.

O. El-Dissouki and W. Huen, "Distributed enumeration on network computers," *IEEE Trans. on Computers*, vol. C-29, no. 9, pp. 818–825, 1980.

M.R. Garey and D.S. Johnson, *COMPUTERS AND INTRACTABILITY A Guide to the Theory of* **NP**-*Completeness*, W.H. Freeman and Company: New York, 1979.

A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, PVM 3 User's Guide and Reference Manual, Oak Ridge National Laboratory: Oak Ridge, Tennessee 37831, USA, May 1993.

P.B. Gibbons, Y. Matias, and V. Ramachandran, "The QRQW PRAM: Accounting for contention in parallel algorithms, in *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, 1994, pp. 638–648. *SIAM J. Comput.*, to appear.

J.J. Hwang, Y.C. Chow, F.D. Anger, and C.Y. Lee, "Scheduling precedence graphs in systems with interprocessor communication times," *SIAM Journal on Computing*, vol. 18, no. 2, pp. 244–257, 1989.

J.K. Lenstra, M. Veldhorst, and B. Veltman, "The complexity of scheduling trees with communication delays," *J. of Algorithms*, vol. 20, pp. 157–173, 1996.

V.M. Lo, Task Assignment in Distributed Systems, Ph.D. thesis, Univ. of Illinois at Urbana-Champaign, USA, Oct. 1983.

V.M. Lo, "Heuristic algorithms for task assignment in distributed systems," *IEEE Trans. on Computers*, vol. 37, no. 11, pp. 1284–1397, 1988.

D.R. Lopez, Models and Algorithms for Task Allocation in a Parallel Environment, Ph.D. thesis, Texas A&M University, Texas, USA, Dec. 1992.

P.R. Ma, E.Y. Lee, and M. Tsuchiya, "A task allocation model for distributed computing systems," *IEEE Trans. on Computers*, vol. C-31, no. 1, pp. 41–47, 1982.

M.G. Norman and P. Thanisch, "Models of machines and computation for mapping in multicomputers," *ACM Computing Surveys*, vol. 25, no. 3, pp. 263–302, 1993.

C. Papadimitriou and M. Yannakakis, "Towards on an architecture-independent analysis of parallel algorithms," *SIAM Journal on Computing*, vol. 19, pp. 322–328, 1990.

H.S. Stone, "Multiprocessor scheduling with the aid of network flow algorithms," *IEEE Trans. on Software Eng.*, vol. SE-3, no. 1, pp. 85–93, 1977.

L.G. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, pp. 103–111, 1990a.

L.G. Valiant, "General purpose parallel architectures," in *Handbook of Theoretical Computer Science*, J. van Leeuwen (Ed.), North Holland, pp. 944–971, 1990b.