PAPER

Scheduling Task In-Trees on Distributed Memory Systems

Sanjeev BASKIYAR[†], *Nonmember*

SUMMARY Tree task structures occur frequently in many applications where parallelization may be desirable. We present a formal treatment of non-preemptively scheduling task trees on distributed memory multiprocessors and show that the fundamental problems of scheduling (i) a task tree in absence of any inter-task communication on a fixed number of processors and (ii) a task tree with inter-task communication on an unbounded number of processors are NP-complete. For task trees that satisfy certain constraints, we present an optimal scheduling algorithm. The algorithm is shown optimal over a wider set of task trees than previous works.

key words: *scheduling, NP-completeness, task-tree, makespan, multiprocessors, interprocessor communication*

1. Introduction

Tree task structures occur frequently in many applications where parallelization may be desirable to improve the execution time. Turner [12] argued that a tree structure is appropriate for large modular programs. The problem of scheduling any directed a-cyclic task graph on a fixed number of processors, to minimize the finish time (makespan) in absence of any inter-task communication was shown to be NP-complete by Ullman [13] and the problem of scheduling any directed a-cyclic task graph, on an unbounded number of processors, where the tasks communicate, was shown to be NP-complete by Papadimitriou and Yannakakis [10]. One may therefore speculate whether optimal algorithms exist for the corresponding problems involving trees. In this paper we put this speculation to rest by providing proofs of NP-completeness for these problems. We also provide an optimal scheduling algorithm for task trees on multiprocessors systems under restricted conditions.

Other related NP-completeness proofs concern minimizing the flow time $\sum_i F_i = \sum_i (f_i - r_i)$ of jobs i in absence of any interprocessor communication times, where f_i and r_i are the finish and ready times of i. Lenstra, Rinnooy Kan and Brucker [9] have shown the problem of non-preemptively scheduling n unit jobs with arbitrary precedence relationships on m processors to minimize flow time to be NP-complete. Sethi [11] has shown the above problem to be NP-complete when the precedence relationships form either an in-tree* or an out-tree. Other results appear in [6] in the section on multiprocessor scheduling. Also, almost exhaustive surveys of parallel machine scheduling research appear in [5] and [8]. Chretienne [4] has attempted to show the problem of scheduling task in-trees to be NP-hard when task duplication on distinct processors is allowed. We disallow task duplication in the problems of this paper. Furthermore, the NP-completeness proof in [4] requires a transformation from an in-tree problem to an out-tree problem. But such a transformation is not applicable in a distributed memory multiprocessor environment. In a task in-tree several tasks output their data to a single successor whenever their processing is complete, if necessary simultaneously. In a task out-tree a parent after completing execution incurs additional time in at least marking which data must be sent to which child**.

Optimal schedules for restricted task trees appear in [1]. In [3], [14] there appears a polynomial time optimal solution for coarse grain task in-trees—a coarse grain tree is defined as one in which the minimum of the execution times of any task in the tree is greater than the maximum of the communication times between any two tasks. The algorithm in [14] is also shown optimal for those single spawn fine grain trees*** in which all tasks and all edges have identical weights. In [2] there is a quadratic time optimal schedule for single level fork/join trees on unbounded number of processors. In this paper we present an optimal scheduling algorithm for task in-trees which satisfy a condition called C^* (explained later in this paper). We show that the scheduling algorithm presented in this paper is optimal for a wider set of task trees, not restricted to coarse grain task in-trees, single spawn trees or single level trees.

The organization of this paper is as follows. In Sect. 2 we define the tree scheduling problem and introduce appropriate notation, in Sect. 3 we prove the

Manuscript received May 11, 2000.

[†]The author is with the Department of Computer Science and Engineering, Auburn University, Auburn, AL 36849, USA.

*An in-tree is a tree that has a unique node called the root node which is the successor of all nodes.

**This time is related to theoretical considerations. Use of multiported memory does not circumvent the problem since marking is still needed to recognize which data item is to be routed to which port. Also, broadcasting all data to all children does not circumvent the problem, in fact, it may change the problem by requiring extra data communication on each edge.

***A single spawn out-tree is a tree in which at most one successor of a non-leaf node can spawn successors; a single merge in-tree is an inverse of a single spawn out-tree.

intractability of scheduling task trees on multiprocessor systems, in Sect. 4 we present an optimal scheduling algorithm for task trees under restricted conditions and in Sect. 5 we summarize our effort.

2. The Scheduling Problem

We consider a directed in-tree task graph consisting of nodes representing tasks and edges representing execution precedences between tasks. A directed edge (i, j) in the task in-tree from node i to node j mandates the execution of node j to begin after the completion of execution of node i. Node i is then said to be a predecessor of node j. A directed edge from i to j may also imply a transfer of an amount d_{ij} of data from i to j.

The target system is a set of identical processors which are completely connected. With each processor is associated an I/O processor that handles the interprocessor communication for that processor such that the execution of the main processor is unaffected by data communication activities. The processor executing node i is represented by P_i.

We are given time mappings for all nodes and edges: $i \to e_i$ and $(i, j) \to c_{ij}$, where e_i represents the execution time of i and c_{ij} represents the time to communicate the output of i to j; if i and j are mapped to distinct processors (i.e. $P_i \neq P_j$), c_{ij} is non-zero otherwise $(P_i \equiv P_j)$, $c_{ij} = 0$. The execution of a task in-tree is complete when the execution of the root node finishes. A schedule defines the assignment of tasks to processors, the start times of tasks and identifies the sending and receiving processors to effect data transfers between two tasks. We are to develop a schedule that minimizes the finish time of the task in-tree on the target system.

In addition to those mentioned above, we will use the following notation throughout the paper to facilitate our analysis. Let

- $T = (V, E)$ represent a task in-tree, V is the set of vertices and E is the set of edges.
- $\mathcal{R}$ represent the set of real numbers and $\mathcal{Z}^+$ represent the set of non-zero positive integers.
- r_i represent the ready time of a node i; i becomes ready for execution as soon as the outputs of the execution of its children are available to it. For leaf nodes $r_i = 0$.
- l represent the number of leaf nodes in a tree.
- s_i represents the time at which the execution of node i begins; $s_i \geq r_i$.
- f_i represent the finish time of i, $f_i = s_i + e_i$.
- $i : P_q$ represent "Assign node i to execute on processor P_q."
- $i \longrightarrow P_q$ represent "Schedule node i to output its data to P_q."
- $i \prec j : P_q$ represent "Assign nodes i and j to execute on processor P_q, such that i executes before j."
- $(r_i)_{j:P_k}$ represent the ready time of node i when node j is assigned to processor P_k.
- $\langle i, j \rangle$ denote that nodes i and j are siblings.
- $\langle i, j, k \rangle$ represent a parent node k with children $\langle i, j \rangle$.
- (i, j) represent a directed edge from node i to node j.
- Let $M = \{1, 2, \ldots, m\}$ represent the set of m completely connected processors; we note that $\forall_{i \in T} P_i \in M \in \mathcal{Z}^+$.
- $\max(a, b) = a$ if $a \geq b$ otherwise b.

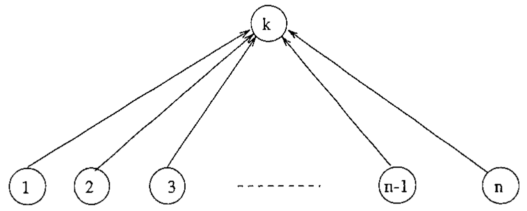


Fig. 1 A task in-tree.

3. Complexity

In this section we present three important scheduling problems and provide NP-completeness proofs for them.

Problem *(Q1)*: Is there a non-preemptive schedule for an in-tree, with integral node execution times and no data communication between any two nodes, using at most m processors, where $1 < m < l$ such that the finish time $t < K$, where $K \in \mathcal{Z}^+$?

We shall show that *(Q1)* is NP-complete.

Proof: The proof follows a reduction from the Bin Packing Problem which is NP-complete. Consider the task in-tree of Fig. 1. The Bin Packing Problem [6] can be stated as follows. Given a finite set U of items each with integral weights, is there a partition of U into disjoint sets $U_1, U_2, \ldots, U_m$ such that the sum of weights of the items in each U_i is K or less? Considering each item of the set U to be a leaf task, and their weights to be node execution times and m the number of processors, the Bin Packing Problem reduces to *Q1*. □

Problem *(Q2)*: Is there a non-preemptive schedule for a task in-tree, with integral task execution times and arbitrary data communication along edges, using at most m processors, where $1 < m < l$, l being the number of leaf nodes in the in-tree such that the finish time $t < K$ where $K \in \mathcal{Z}^+$?

We shall prove that *(Q2)* is NP-complete.

Proof: The proof follows a trivial reduction from *(Q1)*. If in *(Q2)* the communication times along all the edges are set to zero, it becomes *(Q1)* which is NP-complete. □

Problem *(Q3)*: Is there a non-preemptive schedule for an in-tree, with integral node execution and data communication times, on an unbounded number of processors with finish time $t < K$, where $K \in \mathcal{Z}^+$?

We shall prove that *(Q3)* is NP-complete.

Proof: The proof follows a reduction from *(Q2)*. Any optimal schedule, with arbitrary number of processors at hand, must not only examine schedules employing $m \geq l$ processors but also those employing $m < l$ processors, since the latter may yield shorter finish times as shown below.

Let b, e, f, g, h and i be integers representing arbitrary leaf tasks in Fig. 1. Let $r_k^l = \max(r_1+e_1+c_{1k}, r_2+e_2+c_{2k}, \cdots, r_{g-1}+e_{g-1}+c_{g-1\ k}, r_g+e_g, r_{g+1}+e_{g+1}+c_{g+1\ k}, \cdots, r_n+e_n+c_{nk})$ be the ready time of task k when l processors must be employed—tasks g and k when assigned on the same processor give optimal r_k^l. Let $r_k^{l-1} = \max(r_1+e_1+c_{1k}, r_2+e_2+c_{2k}, \cdots, r_{b-1}+e_{b-1}+c_{b-1\ k}, r_b+e_b, r_{b+1}+e_{b+1}+c_{b+1\ k}, \cdots, r_{f-1}+e_{f-1}+c_{f-1\ k}, r_f+e_f, r_{f+1}+e_{f+1}+c_{f+1\ k}, \cdots, r_n+e_n+c_{nk})$ be the ready time of task k when $l-1$ processors must be employed—tasks b, f and k are assigned on the same processor. Let set $S_1 = \{1, 2, \ldots, n\} - \{b, f, k\}$ and $r_p + e_p + c_{pk} = \max_{\forall_{j \in S_1}}(\{r_j + e_j + c_{jk}\})$ where $p \in S_1$. From the above expressions, we deduce that $r_k^{l-1} < r_k^l$ if $r_p + e_p + c_{pk} < \max(r_b + e_b, r_f + e_f, r_g + e_g + c_{gk}) < \max(r_b + e_b + c_{bk}, r_f + e_f + c_{fk}, r_g + e_g)$ or, $\max(r_p + e_p + c_{pk}, r_g + e_g + c_{gk}) < \max(r_b + e_b, r_f + e_f)$ which is feasible [†]. Therefore, a schedule employing $l-1$ processors may yield a shorter finish time than one employing l processors or more (any schedule employing $m > l$ processors can be replaced by one employing $m = l$ processors with equal or shorter finish time). Next, let set $S_2 = \{1, 2, \ldots, n\} - \{e, h, i, k\}$ and $r_q+e_q+c_{qk} = \max_{\forall_{j \in S_2}}(\{r_j+e_j+c_{jk}\})$ where $q \in S_2$. Again, let $r_k^{l-2} = \max(r_1+e_1+c_{1k}, r_2+e_2+c_{2k}, \cdots, r_{e-1}+e_{e-1}+c_{e-1\ k}, r_e+e_e, r_{e+1}+e_{e+1}+c_{e+1\ k}, \cdots, r_{h-1}+e_{h-1}+c_{h-1\ k}, r_h+e_h, r_{h+1}+e_{h+1}+c_{h+1\ k}, \cdots, r_{i-1}+e_{i-1}+c_{i-1\ k}, r_i+e_i, r_{i+1}+e_{i+1}+c_{i+1\ k}, \cdots, r_n+e_n+c_{nk})$ be the ready time of task k when $l-2$ processors must be employed—tasks e, h, i and k are assigned on the same processor. Again, from the above expressions, we observe that $r_k^{l-2} < r_k^l$ if $\max(r_q+e_q+c_{qk}, r_g+e_g+c_{gk}) < \max(r_e+e_e, r_h+e_h, r_i+e_i)$ which is feasible, and therefore a schedule employing $l-2$ processors may give a shorter finish time than one employing l processors. (The feasibility has also been illustrated in Sect. 3.1 by a concrete non-trivial example.) Now, since b, e, f, g, h and i are arbitrary leaf tasks, the above result is extensible to schedules employing $l-3$, $l-4$, $\cdots$, 1 processor(s). By restriction the result applies to arbitrary task in-trees.

Therefore, an algorithm for *Q3* must find an optimal schedule employing $m < l$ processors as well as an optimal schedule employing $m \geq l$ processors. Thus a solution to any instance of *Q2* is available iff a solution to any instance of *Q3* is available. But *Q2* is NP-complete, therefore *Q3* is NP-complete. □

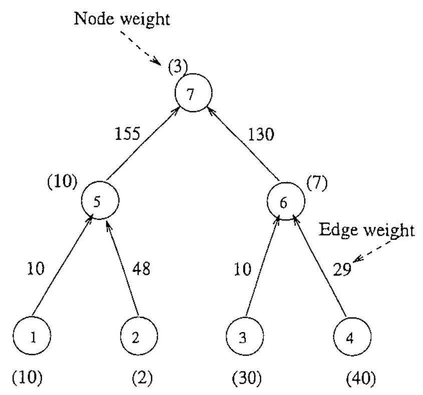


Fig. 2 Example task in-tree.

Furthermore, we observe that the problems in [13] and [10] mentioned earlier, are reducible to problems *(Q1)* and *(Q3)* because:

1. An instance of the problem of scheduling a directed a-cyclic task graph without inter-task communications on a fixed number of processors that is obtained by removing appropriate edges in the graph to form a tree is identical to problem *(Q1)*.
2. An instance of the problem of scheduling a directed a-cyclic graph with inter-task communications on an unbounded number of processors that is obtained by removing appropriate edges of the graph to form a tree is identical to problem *(Q3)*.

3.1 Example

In this section we develop schedules for a non-trivial example task in-tree (of depth 3) shown in Fig. 2. First we develop an optimal schedule, called Schedule A, which is constrained to use $m \geq l$ processors. Next, we develop another schedule, called Schedule A', which can only use $1 < m < l$ processors. Let the finish times using Schedules A and A' be denoted by t and t' respectively.

3.1.1 Schedule A

This schedule must have at least l processors simultaneously active at some time during the course of execution. The properties of a tree guarantee that at any point during the execution of the task in-tree at most l tasks can be ready for execution. Therefore, any schedule that employs more than l processors can be replaced by one that employs l or fewer processors, yielding the same or lower finish time. Therefore, we develop Schedule A using l processors.

[†]The inequality $r_k^{l-1} < r_k^l$ is also satisfied if $\max(r_b + e_b, r_f + e_f, r_g + e_g + c_{gk}) < r_p + e_p + c_{pk} < \max(r_b + e_b + c_{bk}, r_f + e_f + c_{fk}, r_g + e_g)$. Also, assignments to $l-1$ processors, other than the above may be feasible, which give $r_k^{l-1} < r_k^l$.

We assign 1: P_1, 2: P_2, 3: P_3, 4: P_4. Next, we separately examine the schedules in which nodes 5 and 6 execute on the same and different processors.
Case 1: $P_5 \neq P_6$. The finish time will be minimal if we schedule a parent node to execute on a processor that executes one of its children because doing so overlaps the computation with communication along one edge. Since $r_2 + e_2 + c_{25} > r_1 + e_1 + c_{15}$, $5 : P_2, 1 \rightarrow P_2$. Then $r_5 = \max(r_1 + e_1 + c_{15}, r_2 + e_2) = \max(0 + 10 + 10, 0 + 2) = 20$. Since $r_4 + e_4 + c_{46} > r_3 + e_3 + c_{36}$, 6: $P_4, 3 \rightarrow P_4$. Then $r_6 = \max(r_4 + e_4, r_3 + e_3 + c_{36}) = \max(0 + 40, 0 + 30 + 10) = 40$. Since $r_5 + e_5 + c_{57} > r_6 + e_6 + c_{67}$, 7: $P_2, 6 \rightarrow P_2$. Then $r_7 = \max(r_5 + e_5, r_6 + e_6 + c_{67}) = \max(20 + 10, 40 + 7 + 130) = 177$. Also, $f_7 = r_7 + e_7 = 177 + 3 = 180$.
Case 2: $P_5 \equiv P_6$. In an optimal schedule the processor that will execute both nodes 5 and 6 should be one of $P_1, \ldots, P_4$. The following analysis chooses such a processor.
Since $r_2+e_2+c_{25}>r_1+e_1+c_{15}$, $(r_5)_{5:P_2}<(r_5)_{5:P_1}$. (i)
Also, choice of 5: P_1; 5: P_2 does not alter r_6. (ii)
Next, since $r_4 + e_4 + c_{46} > r_3 + e_3 + c_{36}$, $(r_6)_{6:P_4} < (r_6)_{6:P_3}$. (iii)
Also, choice of 6: P_3; 6: P_4 does not alter r_5. (iv)
From results (i)–(iv) we conclude that either 5, 6: P_2 or 5, 6: P_4 will give the minimum r_7. Below we examine the two possibilities.

Case 2.1: 5, 6: P_2. Here, $r_5 = \max(r_1 + e_1 + c_{15}, r_2 + e_2) = \max(0 + 10 + 10, 0 + 2) = 20$. Also, $r_6 = \max(r_4 + e_4 + c_{46}, r_3 + e_3 + c_{36}) = \max(0 + 40 + 29, 0 + 30 + 10) = 69$. We choose $5 \prec 6$: P_2 over $6 \prec 5$: P_2, since it yields the lower finish time of the two. Then $r_7 = \max(r_5 + e_5, r_6) + e_6 = \max(20 + 10, 69) + 7 = 76$. Also, $f_7 = r_7 + e_7 = 76 + 3 = 79$.
Case 2.2: 5, 6: P_4. Here, $r_5 = \max(r_1 + e_1 + c_{15}, r_2 + e_2 + c_{25}) = \max(0 + 10 + 10, 0 + 2 + 48) = 50$. Also, $r_6 = \max(r_3 + e_3 + c_{36}; r_4 + e_4) = \max(0 + 30 + 10, 0 + 40) = 40$. We choose $6 \prec 5$: P_2 over $5 \prec 6$: P_2, as it gives the lower finish time of the two. Then $r_7 = \max(r_6 + e_6, r_5) + e_5 = \max(40 + 7, 50) + 10 = 60$. And, $f_7 = r_7 + e_7 = 63$.

From cases 1, 2.1 and 2.2 we conclude that, $t = 63$ for the optimum schedule employing l processors.

3.1.2 Schedule A'

Let us now consider the following assignment which uses only three processors: P_1, P_2 and P_3.
Assign 1: P_1; $2 \prec 4 \prec 6 \prec 5 \prec 7$: P_2; 3: P_3; $1, 3 \rightarrow P_2$. Then, $r_5 = \max(r_1 + e_1 + c_{15}, r_2 + e_2) = \max(0 + 10 + 10, 0+2) = 20$ and $r_6 = \max(r_3+e_3+c_{36}, \max(f_2, r_4)+e_4) = \max(0 + 30 + 10, \max(2, 0) + 40) = 42$. So, $f_7 = f_4 + e_5 + e_6 + e_7 = 42 + 10 + 7 + 3 = 62$. Thus, $t' = 62$.
We observe[†] from the results in Sects. 3.1.1 and 3.1.2 that $t' < t$.

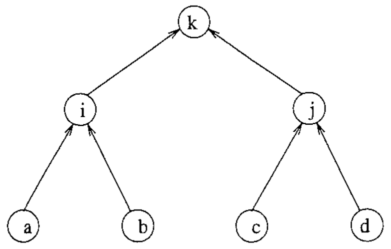


Fig. 3 Task in-tree referred in schedules S and S'.

4. Optimal Schedules for Special Binary Task In-Trees

In this section, we develop an optimal schedule for binary task in-trees that satisfy certain criteria. Let us consider the binary task in-tree shown in Fig. 3. We will proceed by developing two schedules for the subtree $\langle i, j, k\rangle$: Schedule S and Schedule S'. Schedule S is constrained to use distinct processors to execute sibling tasks while Schedule S' is constrained to use the same processor. Let the ready times of any node i under S and S' be represented by r_i and r_i'. First we outline Schedule S.

```
Schedule S
  Assign $i$ and $j$ to distinct processors ($P_i \neq P_j$).
  If $r_i + e_i + c_{ik} \geq r_j + e_j + c_{jk}$ then
    $j \rightarrow P_i$ //Schedule $P_j$ to send its output to $P_i$.
    $k$: $P_i$ //Assign $k$ to execute on $P_i$.
    Then, $r_k = \max(r_i + e_i, r_j + e_j + c_{jk})$ (= $x$, say)
  else
    $i \rightarrow P_j$ //Schedule $P_i$ to send its output to $P_j$.
    $k$: $P_j$ //Assign $k$ to execute on $P_j$.
    Then, $r_k = \max(r_j + e_j, r_i + e_i + c_{ik})$ (= $y$, say)
  endif
end S
```

The constraint in Schedule S guarantees that a processor is available to execute a task as soon as the task is ready for execution, i.e. $\forall_{i \in T}\ r_i = s_i$. It is easy to see that under the condition that sibling tasks must be executed on distinct processors, Schedule S is optimal for the task sub-tree $\langle i, j, k\rangle$. In Schedule S, if the assignment under the condition $r_i+e_i+c_{ik} \geq r_j+e_j+c_{jk}$ were not optimal the only other assignment is the one in the *else* part. But under the condition $r_i + e_i + c_{ik} \geq r_j+e_j+c_{jk}$, $x = \max(r_i+e_i, r_j+e_j+c_{jk}) \leq r_i+e_i+c_{ik}$ and $y = \max(r_j + e_j, r_i + e_i + c_{ik}) \geq x$. Therefore, the assignment in the *if* part gives a shorter schedule length than the *else* part. The proof for the optimality of the *else* part is symmetrical.
In Fig. 4, we present Procedure *MinMakespan*

[†]This is also true for task trees other than the one in Fig. 2. As we will observe later, many task trees within the set of task trees for which Condition C is not satisfied, fall within this category.

which is based upon Schedule S. Under the requirement that distinct processors be used to execute sibling tasks, Procedure *MinMakespan* optimally schedules any binary task in-tree in $O(n)$ time, where n is the number of nodes in the task tree.

Let us now consider Schedule S'. Below we outline Schedule S'.

```
Schedule $S'$
  //Assign all of $i, j$ and $k$ to a single processor.
  If $r_i' \le r_j'$ then
     $i \prec j \prec k$: $P_i$   //Schedule $i$ to execute before $j$
     Then, $r_k' = \max(s_i' + e_i, r_j') + e_j$
  else
     $j \prec i \prec k$: $P_i$   //Schedule $j$ to execute before $i$.
     Then, $r_k' = \max(s_j' + e_j, r_i') + e_i$
  endif
end $S'$
```

Under the condition that sibling tasks must be executed on the same processor, Schedule S is optimal for the task sub-tree $\langle i, j, k\rangle$ since it considers all possible assignments. In Schedule S', unlike in Schedule S, we use start times s_i' and s_j' instead of ready times r_i' and r_j', since it is quite possible that $s_i' > r_i'$. As an example, when a single processor executes more than one leaf node f, $s_f' \ge r_f'$ since $r_f = 0$ for leaf f. As another example, consider Fig. 3 in which if i were assigned to the processor executing c, namely P_c, and the executions of a and b and their subsequent outputs of data to P_c complete before the execution of node c completes, then $s_i' > r_i'$.

Now, if for every node k, of a tree, $r_k \le r_k'$ (hereafter called Condition C), Schedule S is optimal. Therefore, *MinMakespan* is optimal for task trees satisfying Condition C. Now, using *MinMakespan* we can determine $\forall_{k \in T}\, r_k$ polynmially but we can not evaluate $\forall_{k \in T}\, r_k'$ in polynomial time—if we could, *(Q3)* would not be NP-complete. Therefore, we can not determine in polynomial time whether C holds. The following discussion introduces a new Condition C^* which can be evaluated in polynomial time.

We observe,

$$r_k' = \min(\max(s_i' + e_i, r_j') + e_j, \max(s_j' + e_j, r_i') + e_i)$$

$$\ge \min(\overbrace{\max(r_i' + e_i, r_j') + e_j}^{E}, \overbrace{\max(r_j' + e_j, r_i') + e_i}^{F}),$$

$$\text{since } \forall_{i \in T} s_i' \ge r_i' \qquad \text{(i)}$$

$$\text{Let } R_k = \min(\overbrace{\max(r_i + e_i, r_j) + e_j}^{G}, \overbrace{\max(r_j + e_j, r_i) + e_i}^{H}) \qquad \text{(ii)}$$

Let $\forall_{i \in T} r_i \le R_i$ be called Condition C^*. Below we prove by induction that if C^* holds, C holds.

For any leaf $i \in T$, $r_i \le r_i'$ is satisfied because the ready times of leaf tasks are zero in both Schedules S and S'.

Next, for siblings i and j of any subtree $\langle i, j, k\rangle$, if $r_i \le r_i'$ and $r_j \le r_j'$, we show that for the parent node k, $r_k \le r_k'$ too. Using $r_i \le r_i'$ and $r_j \le r_j'$ in Eqs. (i) and (ii) we get, $G \le E$, $H \le F$. Using *Lemma 1* (below) in Eqs. (i) and (ii) we get $R_k \le r_k' \cdots$ (iii). From Condition C^* for node k, we have $r_k \le R_k \cdots$ (iv). From (iii) and (iv) we have, $r_k \le r_k'$.

Therefore by induction $\forall_{i \in T} r_i \le r_i'$, which is Condition C. □

Thus, for task in-trees for which C^* holds, *MinMakespan* provides an optimal schedule in $O(n)$ time.

Lemma 1: If $p = \min(E, F)$, $q = \min(G, H)$ and $G \le E$, $H \le F$ then $q \le p$.

Proof: Consider the case when $E \le F$.
We have $p = E$.
If $G \le H, q = G \Rightarrow q \le E \Rightarrow q \le p$.
If $G \ge H, q = H \Rightarrow q \le F \Rightarrow q \le E \Rightarrow q \le p$

The proof for $E > F$ parallels that for $E \le F$. □

Lemma 2: Condition C^* holds if $\max_{\forall (i,j) \in E} c_{ij} \le$

```
Procedure MinMakespan
//Inputs: $T$, a task in-tree as described before. A set of processors $M = \{1, 2, \ldots, m\}$, ready times $r_i$ for all leaf nodes $i$.
//Definitions: Functions LeftChild, RightChild operate on any $k \in T$, such that LeftChild(k) = left child node of $k$ if one exists, otherwise nil. RightChild(k) = right child node of $k$ if one exists, otherwise nil. Leaf(k) iff LeftChild(k) = RightChild(k) = nil.
//Outputs: (i) $\forall_{i \in T}\ P : i \rightarrow M$ such that $P_i$ represents the processor to which node $i$ is assigned (ii) $\forall_{i \in T}\ s_i$ (iii) A schedule of interprocessor communications.

  static $n \leftarrow 0$
  if $i \ne nil$ then
     $i \leftarrow$ LeftChild(i)
     $j \leftarrow$ RightChild(i)
     MinMakespan(i)
     MinMakespan(j)
     if Leaf(i) then
        $n \leftarrow n + 1, P_i \leftarrow n$
     else
        if $r_i + e_i + c_{ik} \ge r_j + e_j + c_{jk}$ then
           $j \rightarrow P_i$   //Schedule $P_j$ to send its output to $P_i$.
           $k : P_i$   //Assign $k$ to execute on $P_i$.
           Then, $r_k = \max(r_i + e_i, r_j + e_j + c_{jk})$
        else
           $i \rightarrow P_j$   //Schedule $P_i$ to send its output to $P_j$.
           $k : P_j$   //Assign $k$ to execute on $P_j$.
           Then, $r_k = \max(r_j + e_j, r_i + e_i + c_{ik})$
        endif
     endif
  endif
end MinMakespan
```

Fig. 4 Procedure *MinMakespan*.

$\min_{\forall i \in V} e_i$.

Proof: We have, $R_k = \min(\max(r_i + e_i, r_j) + e_j, \max(r_j + e_j, r_i) + e_i)$. We show that when $r_i + e_i + c_{ik} \geq r_j + e_j + c_{jk}$ (say, condition (i)) C^* holds. Under condition (i), from Schedule S we have, $r_k = \max(r_i + e_i, r_j + e_j + c_{jk}) \cdots$ Eq. (i). Using $e_j \geq c_{jk}$ in condition (ii), we have $r_i + e_i + e_j \geq r_j + e_j + c_{jk}$ (say, condition (ii)). Therefore, $R_k = \min(r_i + e_i + e_j, \max(r_j + e_i + e_j, r_i + e_i)) \cdots$ Eq. (ii). The first term of Eq. (ii), $r_i + e_i + e_j$ is greater than the first term of Eq. (i) and it is also greater than or equal to the second term of Eq. (i) by virtue of condition (ii). Again, the second term of Eq. (ii) is greater than r_k since using $e_i \geq c_{jk}$ in Eq. (i) we have $r_k \leq \max(r_i + e_i, r_j + e_j + e_i)$. Therefore, $R_k \geq r_k$.

By symmetry, the proof when $r_j + e_j + c_{jk} > r_i + e_i + c_{ik}$ parallels the proof above. □

Lemma 3: Schedule S is optimal for binary coarse grain trees.

Proof: In a coarse grain tree, $\max_{\forall(i,j) \in E} c_{ij} < \min_{\forall i \in V} e_i$. The condition in *Lemma 2* includes the above condition. Therefore, Schedule S is optimal for coarse grain trees. □

We now show, using the example below, that C^* holds for some fine grain trees too.

Example 1: Consider the task in-tree in Fig. 5 which is not coarse grain. In this task in-tree, the minimum task execution time of 2.5 is less than the maximum inter-task communication time of 3.5. Below we show that Condition C^* holds for this tree. We assign nodes 4, 5, 6 and 7 on distinct processors called P_4, P_5, P_6, and P_7. We have $r_4 = R_4 = r_5 = R_5 = r_6 = R_6 = r_7 = R_7 = 0$.

Now, since $r_4 + e_4 + c_{42} < r_5 + e_5 + c_{52}$ (i.e. $0 + 2.5 + 2.1 < 0 + 3.9 + 2.2$), we assign 2: P_5 and $4 \rightarrow P_5$. Then, $r_2 = \max(r_5 + e_5, r_4 + e_4 + c_{42}) = \max(0 + 3.9, 0 + 2.5 + 2.1) = 4.6$. Also, $R_2 = \min(\max(r_4 + e_4, r_5) + e_5, \max(r_5 + e_5, r_4) + e_4) = e_4 + e_5 = 2.5 + 3.9 = 6.4$. We observe, $r_2 < R_2$.

Again, since $r_6 + e_6 + c_{63} > r_7 + e_7 + c_{73}$ (i.e. $0 + 4.1 + 3.5 > 0 + 3.1 + 2.1$), we schedule 3: $P_6, 7 \rightarrow P_6$. Then, $r_3 = \max(r_6 + e_6, r_7 + e_7 + c_{73}) = \max(0 + 4.1, 0 + 3.1 + 2.1) = 5.2$. Also, $R_3 = \min(\max(r_6 + e_6, r_7) + e_7, \max(r_7 + e_7, r_6) + e_6) = 4.1 + 3.1 = 7.2$. Here too, $r_3 < R_3$.

Since $r_2 + e_2 + c_{21} < r_3 + e_3 + c_{31}$ (i.e. $4.6 + 3.1 + 1.9 < 5.2 + 4.9 + 2.9$), we schedule 1: $P_3, 2 \rightarrow P_3$. Then, $r_1 = \max(r_3 + e_3, r_2 + e_2 + c_{21}) = \max(5.2 + 4.9, 4.6 + 3.1 + 1.9) = 10.1$. Also, $R_1 = \min(\max(r_2 + e_2, r_3) + e_3, \max(r_3 + e_3, r_2) + e_2) = \min(\max(4.6 + 3.1, 5.2) + 4.9, \max(5.2 + 4.9, 4.6) + 3.1) = 12.6$. Clearly, $r_1 < R_1$.

Therefore, we observe that for all nodes i of the task in-tree, $r_i \leq R_i$. Therefore C^* is satisfied for the task in-tree of Fig. 5. □

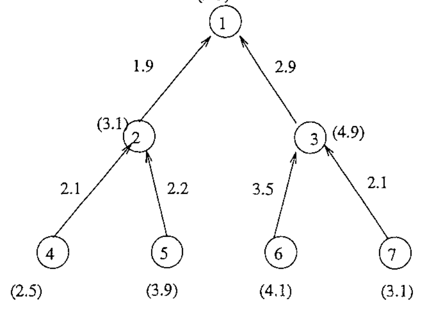


Fig. 5 Example fine grain task in-tree.

An example where the condition $\max_{\forall(i,j) \in E} c_{ij} \leq \min_{\forall i \in V} e_i$ in *Lemma 2* holds is a task in-tree whose node execution times are non-zero positive integers and the inter-task communication times are either zero or unity, i.e. $\forall_{i \in T}\, e_i \in \mathcal{Z}^+$ and $\forall_{(i,j) \in T}\, c_{ij} = 0$ or 1. Although restrictive, problems satisfying the above criteria may not be that uncommon. For instance, the communication times can be constant over all edges if the data communicated is small, particularly in message passing multiprocessors when the times to initiate and terminate a communication may exceed the actual data transfer time and more so if data is sent in packets. Thus, the communication time upto a certain amount of data transmission can be constant. Furthermore, in many instances the node execution times and the data communication times can be scaled to satisfy the above criteria.

5. Conclusion

We have shown three important scheduling problems to be NP-complete. We have also developed an optimal schedule for restricted task in-trees on completely connected l processor systems, where l is the number of leaf nodes of the task tree. This schedule is not only optimal for coarse grain trees, but for all trees which satisfy condition C^*. The scheduling algorithm has been shown to be optimal over a wider set of task trees than previous works. The problem *(Q1)* is more fundamental than the problem in [13] mentioned earlier because with progress of execution in the task in-tree the number of ready tasks decreases and gets closer to the number of processors available, such may not be the case in a directed a-cyclic task graph.

References

[1] S. Baskiyar, "Architectural and scheduler support for object-oriented programs," Ph.D. Thesis, University of Minnesota, 1993.

[2] P. Chretienne, "Task scheduling over distributed memory machines," Proc. International Workshop on Parallel and Distributed Algorithms, North Holland, 1989.

[3] P. Chretienne, "A polynomial algorithm to optimally schedule tasks on a virtual ideal dtributed system under tree-like precedence constraints," European J. Operational Research, vol.43, no.2, pp.225–230, 1989.

[4] P. Chretienne, "Tree scheduling with communication delays," Discrete Applied Mathematics, vol.49, no.1–3, pp.129–141, 1994.

[5] T.C.E. Cheng and C.C.S. Sin, "A state-of-the-art review of parallel machine scheduling research," European J. Operations Research, North-Holland, 1990.

[6] M.R. Garey and D.S. Johnson, Computers and Intractability, W.H. Freeman and Company, pp 238–241, 1991.

[7] A. Gerasoulis, S. Venugopal, and T. Yang, "Clustering task graphs for message passing architectures," Proc. ACM Int'l. Conf. on Supercomputing, pp.447–456, 1990.

[8] E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan, "Recent developments in deterministic sequencing and scheduling," A Survey in Deterministic and Stochastic Scheduling, eds. M.A.H. Dempster et al., 1982.

[9] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker, "Complexity of machine scheduling problems," Annals of discrete mathematics, vol.1, pp.343–362, 1977.

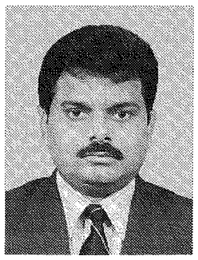
[10] C. Papadimitriou and M. Yannakakis, "Towards an architecture-independent analysis of parallel algorithms," SIAM J. Computing, vol.19, 1990.

[11] R. Sethi, "On the complexity of mean flow time scheduling," Mathematics of Operations Research, vol.2, pp.320–330, 1977.

[12] J. Turner, "The structure of modular programs," Commun. Ass. Comput. Mach., vol.SE-4, pp.254–258, May 1978.

[13] J. Ullman, "NP-complete scheduling problems," J. Computer System and Sciences, vol.10, pp.384–393, 1975.

[14] T. Yang and A. Gerasoulis, "DSC: Scheduling parallel tasks on an unbounded number of processors," IEEE Trans. Parallel & Distributed Systems, vol.5, no.9, pp.951–967, 1994.



Sanjeev Baskiyar received the B.S. degree in Electronics and Communication Engineering from the Indian Institute of Science, Bangalore and the M.S.E.E. and Ph.D. degrees from the University of Minnesota, Minneapolis. Currently he is Assistant Professor in the Department of Computer Science and Engineering at Auburn University, Auburn, AL. His work experience includes working as an Assistant Professor at Western Michigan University, as a Senior Software Engineer in the Unisys Corporation and as a Computer Engineer in the Tata Engineering and Locomotive Company, Jamshedpur, India. His current research interests are in Task Mapping onto Multiprocessors, Computer Systems Architecture and Real-time and Embedded Computing.