

Scheduling Independent Tasks with Due Times on a Uniform Processor System

SARTAJ SAHNI AND YOOKUN CHO

University of Minnesota, Minneapolis, Minnesota

ABSTRACT. An algorithm to preemptively schedule n tasks on m uniform processors is presented. It is assumed that each task is available at time 0. Associated with each task is a due time by which it is to be completed. The algorithm schedules all tasks to complete by their due times whenever possible. The asymptotic time complexity of the algorithm is $O(n \log n + mn)$. It generates $O(mn)$ preemptions in the worst case. An example of n tasks requiring $O(mn)$ preemptions is also presented. The algorithm can also be used when all tasks have the same due times but different release times.

KEY WORDS AND PHRASES: independent tasks, preemptive schedule, due time, uniform processors, complexity

CR CATEGORIES: 5.25, 5.3, 5.4

1. Introduction

Let $P = \{P_1, P_2, \dots, P_m\}$ be a set of m processors. Let $t_i, r_i,$ and $d_i, 1 \leq i \leq n,$ be the task times, release times, and due times, respectively, of n independent tasks. Associated with each processor P_i is a speed $s_i, s_i > 0.$ Processor P_i has an effective processing capability of s_i units of processing per time unit. Task j can be processed on P_i in t_j/s_i units. The processors are said to be *uniform*, as they operate at a constant speed independent of time. When $s_i = 1, 1 \leq i \leq m,$ the processors are said to be *identical*. In this paper we are concerned only with preemptive schedules. A schedule S for the n tasks is a *DD-schedule* iff the processing of each task commences no earlier than its release time and completes no later than its due time. A DD-schedule will also be referred to as a *feasible* schedule.

For the case of identical processors, Horn [3] presents an $O(n^3)$ algorithm to obtain a DD-schedule for any set of n independent tasks for which such a schedule exists. Sahni [5] presents a fast algorithm that obtains DD-schedules (whenever they exist) when all tasks have either the same release time or the same due time. For the case of two uniform processors, Bruno and Gonzalez [1] present an $O(n^3)$ algorithm that obtains a DD-schedule (whenever one exists). Gonzalez and Sahni [2] have developed an $O(n + m \log m)$ algorithm that can be used when all tasks have the same release time and also the same due time.

In this paper we study the case when all tasks have the same release time. Different tasks may, however, have different due times. Our algorithm to construct a DD-schedule (if one exists) for this case takes $O(mn)$ time in the worst case. DD-schedules containing at most mn preemptions are generated. While the algorithm is discussed in terms of a set of

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

This research was supported in part by the National Science Foundation under Grants MCS 76-21024 and MCS 78-15455.

Authors' present addresses: S. Sahni, Department of Computer Science, University of Minnesota, 136 Lind Hall, 207 Church St. SE, Minneapolis, MN 55455; Y. Cho, Computer Science Department, Seoul National University, Seoul, Korea.

© 1980 ACM 0004-5411/80/0700-0550 \$00.75

tasks with the same release time, it should be noted that the algorithm may also be used when all tasks have the same due time but different release times. As pointed out in [3] and [5], the two situations are isomorphic. An instance of one may be transformed into an instance of the other by simply interchanging the roles of due times and release times.

2. Preliminaries

In this section we develop an algorithm that constructs a DD-schedule (if one exists) for n independent tasks all of which have the same release time and the same due time. The processor model assumed is, however, more general than the uniform processor model. In the next section this algorithm will be used to construct a DD-schedule (if one exists) for the one-release-time-many-due-time problem for a uniform processor system.

A *generalized processor* is a processor whose speed is a nondecreasing function of time. While the ideas presented in this section can be applied when the processor speed is given by any nondecreasing function of time, we limit ourselves here to the case where the speed changes only a finite number of times. Thus the characteristics of any generalized processor G in the time interval $[0, d]$ may be described by a finite list of pairs (σ_i, γ_i) , $1 \leq i \leq p$. σ_i and γ_i , respectively, represent time and speed. G operates at speed γ_i in the interval $[\sigma_i, \sigma_{i+1}]$. We may assume that $\sigma_1 = 0$, $\sigma_{p+1} = d$, $\sigma_i < \sigma_{i+1}$, $\gamma_i \geq 0$, and $\gamma_i < \gamma_{i+1}$. The *effective processing capability*, L , of G in the interval $[0, d]$ is equal to $\sum_{i=1}^p (\sigma_{i+1} - \sigma_i) \gamma_i$.

A *generalized processor system* (GPS) is an ordered set of m generalized processors $\{G_1, G_2, \dots, G_m\}$, $m \geq 1$. This set of generalized processors has the property that at each instance $t \in [0, d]$, the speed of G_i is no less than that of G_{i+1} , $1 \leq i < m$.

It should be pointed out that a uniform processor system in which the processors have been ordered by speed is a special case of a GPS. The algorithm we shall develop here for a GPS is different from that developed in [2] for uniform processor systems. Theorem 1 gives a necessary and sufficient condition that every set of n tasks must satisfy if they are to be completed in the interval $[0, d]$ on a GPS. The remainder of this section is devoted to the proof of this theorem. Since the proof is constructive, it immediately leads to an algorithm for obtaining a DD-schedule for the case when all tasks have the same release time and the same due time. From now on we shall refer to a generalized processor simply as a processor. This should lead to no confusion.

THEOREM 1. Let $\{G_1, G_2, \dots, G_m\}$ be an m processor GPS. Let t_i , $1 \leq i \leq n$, be the task times of any set of n independent tasks. Assume $t_i \geq t_{i+1}$, $1 \leq i < n$, and that $n \geq m$. Let $T_i = \sum_{j=1}^i t_j$, $1 \leq i < m$, and $T_m = \sum_{j=1}^m t_j$. Let L_i be the effective processing capability of G_i in the interval $[0, d]$, $1 \leq i \leq m$. The n tasks can be scheduled to complete by time d iff $\max_i \{T_i / \sum_{j=1}^i L_j\} \leq 1$.

The statement of the theorem assumes that $n \geq m$. In case this is not true (i.e., $n < m$) we may discard G_{n+1}, \dots, G_m from the GPS as there is no advantage to scheduling a task on any of these processors. This follows from the following observations: (i) a task may be scheduled on at most one processor at any time, and (ii) at any instance $t \in [0, d]$, each of processors G_1, \dots, G_n is no slower than any of G_{n+1}, \dots, G_m .

The "only if" part of the theorem is easily established. We need to show that if $\max\{T_i / \sum_{j=1}^i L_j\} > 1$, then the set of tasks cannot be completed in the interval $[0, d]$. If we consider only the largest i tasks, $i < m$, then for the same reasons as above we need consider only G_1, \dots, G_i . If $T_i > \sum_{j=1}^i L_j$, then these i processors do not have enough effective processing capability in $[0, d]$ to complete these tasks. It should be easy to see that if this is the case, then these i tasks cannot be completed in $[0, d]$ when considered together with the remaining $n - i$ tasks and $m - i$ processors. Finally, if $T_m > \sum_{j=1}^m L_j$, then there is not enough effective processing capability in $[0, d]$ to perform all n tasks.

The proof for the "if" part is by construction. We show how to obtain a feasible schedule when $\max_i \{T_i / \sum_{j=1}^i L_j\} \leq 1$. This construction requires the notion of a disjoint processor system (DPS). A processor G is said to be *idle in the interval* $[t_1, t_2]$ if it operates at nonzero

