Scheduling on two Uniform Processors with Resource Constraints to Minimize Mean Flow Time

Wei-Kuan Shih and Huei-Long Wang

Department of Computer Science,
National Tsing Hua University, HsinChu, Taiwan, R.O.C.
Email:{wshih, wyvern}@cs.nthuu.edu.tw

ABSTRACT

In this paper, we study the problem of scheduling a set of tasks to minimize mean flow time. The tasks have equal processing time and are scheduled on two uniform processors. The faster processor is *b* times faster than the slower one. The schedule needs to satisfy the resource constraints, that is, there are *r* units of resources, all are of the same kind. At any instant, the total number of resources held by the tasks executed on both uniform processors is less than or equal to *r*. Tasks are not preemptive. We propose an $O(n^2)$ algorithm to find an asymptotically optimal schedule for this problem. Our algorithm can be extended to develop a good approximation algorithm for a multiprocessor system with arbitrary number of processors.

1.INTRODUCTION

This paper addresses the following scheduling problem. We are given a set of independent tasks $\mathbf{T} = \{T_1, T_2, \ldots T_n\}$. The tasks are to be executed non-preemptively on a system that contains: (1) two uniform processors, the processor P_1 is *b* times faster than the slower processor P_2 , and (2) *r* units of resources, all are of the same kind. The processing time of every task on the faster processor P_1 is 1. Therefore, it takes *b* units of time to complete the task on the slower processor. In addition to requiring a processor, each task T_i requires $R(T_i)$ units of resources to execute. Hereafter, by a task set, we mean specifically a set of tasks with the characteristics described above unless otherwise states.

The goal is to schedule the tasks so that their mean flow time is minimized. Specifically, a schedule σ is an assignment of the tasks in **T** to the processors P_1 and P_2. Tasks assigned to the same processor must be executed in disjointed intervals of time. This assignment satisfies the following constraints:

(1) Each task is executed by one processor.

(2) Each processor executes at most one task at a time.

(3) The starting time $S(T_i)$ of T_i is the instant at which T_i begins its execution in σ. Its completion time $C(T_i)$ is $S(T_i) + 1$ if T_i is executed on P_1 and is $S(T_i) + b$ if T_i is executed on P_2.

(4) At any instant, the total number of resources held by the tasks executed on P_1 and P_2 is less than or equal to *r*.

The condition in (4) is referred to as the *resource constraint*. Given a schedule σ and the set of completion times $C(T_j)$ of the *n* tasks in **T**, the *mean flow time* of the schedule is

$$F_n = \sum_{i=1}^{n} C(T_i) / n$$

An optimal schedule of the tasks in **T** on processors P_1 and P_2 is one that meets the resources constraints and has the minimum mean flow time.

The problem of finding such optimal schedules was suggested by Blazewicz, et al [1] as an open problem. In this paper, we proposed an $O(n^2)$ algorithm that finds asymptotically optimal schedules; a schedule is said to be *asymptotically optimal* if its mean flow time is minimum as the number of tasks *n* approaches infinity. In the special case where the speed ratio *b* is an integer, our algorithm can be simplified to find optimal schedules.

The problems of scheduling with resource constraints have been studied extensively. Excellent surveys on complexity analysis of these problems can be found in [2-5]. The problem of scheduling non-preemptive tasks with arbitrary processing times and resource requirements to minimize mean flow time is shown in [1] to be NP-hard even for the case of two identical processors. An $O(n^3)$ algorithm is given in [1] for finding schedules with minimum mean flow time of independent, non-preemptive tasks, each of which has arbitrary processing time and requires 1 unit of resource of the same type.

The remaining part of this paper is organized as follows: Section 2 describes the characteristics of asymptotically optimal schedules. Section 3 gives an $O(n^2)$ algorithm that finds asymptotically optimal schedules when the speed ratio *b* is arbitrary. Section 4 is the conclusion and future works.

2.THE PROPERTIES OF ASYMPTOTICALLY OPTIMAL SCHEDULES

Let **L** denotes a subset of *l* tasks in **T** whose resource requirements are so large that they cannot be scheduled with other tasks at the same time. In any optimal schedule, tasks in **L** are scheduled in the faster processor P_1, leaving the

slower processor P_2 in idle. We can find an asymptotically optimal schedule of **T** by first finding an asymptotically optimal schedule of the tasks in **T** - **L** and then scheduling the tasks in **L** on P_1 either before or after all tasks in **T** - **L** depending on the value of *l*. Without loss of generality, we will confine our discussion hereafter to the case where the subset **L** is empty.

We use σ_o and σ_a to denote an optimal schedule and an asymptotically optimal schedule, respectively. When it is necessary to distinguish the starting time and completion time of a task T_j in a particular Schedule σ, we denote the starting time and completion time of task T_j by $S(T_j,\sigma)$ and $C(T_j,\sigma)$, respectively. For any speed ratio, we can partition the time interval into sufficiently small time slices such that every task starts and ends its execution at the boundaries of some time slices in any schedule. We measure the lengths of all time intervals in terms of numbers of time slices. In terms of the time slices, the starting times and completion times of all tasks are integers. We say that a task T_i is executed later (or earlier) than T_j in a schedule σ if $S(T_i,\sigma) \geq S(T_j,\sigma)$ (or $S(T_i,\sigma) < S(T_j,\sigma)$), that is, the starting time of T_i is no earlier than the starting time of T_j. Finally, we denote the number of tasks assigned to the processor P_i according to the schedule σ by $N(i,\sigma)$ for i = 1 and 2, and the completion time of the last task on processors P_1 or P_2 in the schedule σ by $f(\sigma)$.

To characterize asymptotically optimal schedules, we need the following definitions:

Definitions: Given a schedule σ_n, let $T_1,T_2,...,T_k$ be the tasks executed on processor P_1 and $T_{k+1},T_{k+2},..., T_n$ be the tasks executed on processors P_2.

(1) The schedule σ_n is said to be *normalized* if the resource requirements of the tasks are such that $R(T_i) \geq R(T_j)$, here $1 \leq i \leq k$ and $k + 1 \leq j \leq n$.

(2) Suppose that the tasks are indexed so that $R(T_1) \geq R(T_2) \geq ... \geq R(T_k) \geq R(T_{k+1}) \geq ... \geq R(T_{n-1}) \geq R(T_n)$. A normalized schedule is said to be *compact* if in it (a) the starting times of the tasks are such that $S(T_1) \geq S(T_2) \geq ... \geq S(T_k)$ and $S(T_{k+1}) \leq S(T_{k+2}) \leq ... \leq S(T_n)$ and (b) the slower processor P_2 is busy during the interval $(0, b(n-k))$ (i.e., $S(T_{k+1}) = 0$ and $C(T_n) = b(n-k)$), and (c) the faster processor P_1 is busy during the interval $(S(T_k), S(T_k) + k)$ (i.e., $C(T_1) = S(T_k) + k$).

Figure 1 illustrates a compact schedule. Tasks T_1 to T_{12} are scheduled on P_1. Tasks T_{13} to T_{19} are scheduled on P_2. The tasks scheduled on the slow processor P_2 start at time 0 and never idle after the processor starts its execution. The fast processor P_1 has some initial idle time, and never stops executing tasks until all tasks assigned to P_1 are finished.

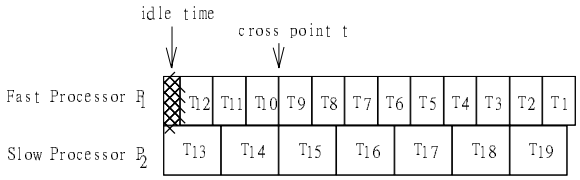

Figure 1. An example illustrates a compact schedule and a cross point.

(3) We refer to a time instant *t* in a schedule σ at which there are a task T_i starting on P_1 and a task T_j starting on P_2 as a *cross point*. That is, $S(T_i,\sigma) = S(T_j,\sigma) = t$ for some T_i on P_1 and T_j on P_2. For example, in Figure 1 task T_9 and T_{15} start at the same time *t*. Therefore, *t* is a cross point.

Our algorithm uses the following strategy to finding asymptotically optimal schedule. We at first find a compact schedule σ_c. After a compact schedule σ_c is found, we rearrange the execution order of some tasks to produce an asymptotically optimal schedule. To prove that our strategy can find an asymptotically optimal schedule, we need the following lemmas.

Lemma 1: Given an optimal schedule σ_o with the number of tasks assigned to processor P_1 being *k*, we can find from σ_o a compact schedule σ_c in which the number of tasks assigned to P_1 is also equal to *k*, and the length of the idle period from 0 to $S(T_k)$ on P_1 is at most equal to the total length *I* of time intervals during which P_1 is idle in σ_o. In other words, $N(1,\sigma_c) = N(1,\sigma_o) = k$, and $S(T_k,\sigma_c) \leq I$.

Proof: Given an optimal schedule σ_o, we sort all tasks assigned to the slow processor P_2 in non-increasing order according to their resource requirements; the tasks with larger resources requirements are executed earlier. Moreover, the *n-k* tasks are assigned to the interval from 0 to *b*(*n-k*) on P_2. During the sorting, the idle periods on the slow processor P_2 will be moved to the end of the schedule. The tasks assigned to the fast processor P_1 will be moved to keep the resource constraints satisfied. Tasks on the fast processor P_1 may be preempted. In other word, when a task on P_2 is moved, the task slices on P_1 which are scheduled in the same time interval in σ_o are moved also. Therefore, the resource constraints remain satisfied after the sorting is finished. Tasks on P_1 are split into many task slices and are assigned to uncontiguous time intervals.

Figure 2 illustrates how the sorting works. In this figure, the slow processor P_2 has three tasks and $R(T_2) \geq R(T_5) \geq R(T_7)$. We sort task T_2, T_5 and T_7 according to their resource requirements. During the sorting, task T_2 is moved to the interval starting at time 0. To satisfy the resource constraints, task T_4 and part of task T_6 need to be moved to the same time slot on P_1. Tasks T_6 is divided into two pieces during the sorting.

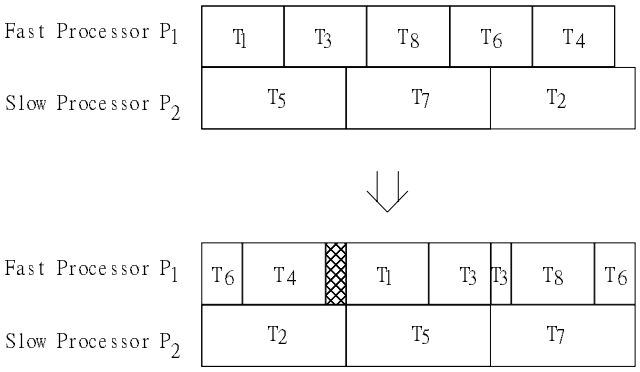

Figure 2. The result of sorting tasks on P_2

In order to put the task slices on P_1 back together (i.e., in contiguous time intervals), we sort all task slices on P_1 according to their resource requirements; the tasks with smaller resource requirements are executed earlier. Since all tasks on processor P_2 are already sorted, sorting task slices on P_1 will not violate the resource constraints. The example in Figure 3 illustrates how to put the task slices on P_1 together without violating the resource constraints. In this example, $R(T_1) \geq R(T_3) \geq R(T_4) \geq R(T_6) \geq R(T_8)$. Let's look at two tasks--T_8 and T_6 that violate the sorting order. Task T_8 and task T_6 need to be swapped. Since $R(T_2) \geq R(T_7)$, therefore, $r \geq R(T_6) + R(T_2) \geq R(T_6) + R(T_7)$. It means that T_6 and T_7 can be scheduled in the same time. Therefore, moving T_6 to the time interval originally assigned to T_8 will not violate the resource constraints. Moreover, since $R(T_6) \geq R(T_8)$, we can move T_8 to the time interval assigned to T_6 without violating the resource constraints. The sorting process of the tasks on the fast processor P_1 can be done by applying the same swapping method repeatedly to tasks that violate the sorting order.

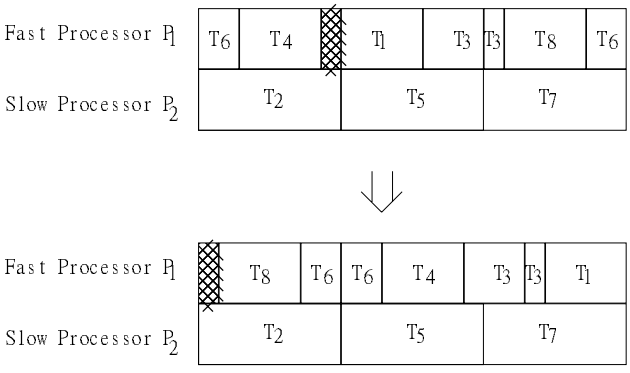

Figure 3. The result of sorting tasks on P_1

After the sorting is finished, we assume that the first tasks assigned on P_1 and on P_2 are T_a and T_b, respectively. If $R(T_a) < R(T_b)$, we can swap T_a and T_b. The task swapping process will be illustrated in Figure 4. The compact schedule is constructed by iterations. Each iteration has two steps. In the first step, we do the sorting process. In the second step, we swap the task if necessary. The iterations are repeated until no tasks need to be swapped. Figure 4 illustrates the change of the schedule after doing one iteration. In this example, the first task assigned to processor P_1 (or P_2) is T_8 (or T_2). Since $R(T_8) \leq R(T_2)$, we need to swap these two tasks to make the schedule compact. Since T_6 is the task scheduled at the same time with T_2, we need to check if the resource constraint is violated after we swap task T_8 and T_2. Since $R(T_8) + R(T_6) \leq R(T_2) + R(T_6) \leq r$, we can schedule T_8 and T_6 at the same time. By using the iterations of sorting and swapping, we eventually will find a compact schedule.

It is always possible to make the starting time $S(T_k, \sigma_c)$ of the first task T_k on P_1 sufficiently late so that P_1 once becoming busy remains busy until the end of the schedule. The new schedule thus produced is a compact schedule. To find an upper bound of $S(T_k, \sigma_c)$ so that the schedule is compact, we note that after the task slices on P_1 and P_2 are sorted as described above, the total length I of all intervals between 0 and $b(n\text{-}k)$ during which P_1 is idle is less than or equal to the total length of time intervals during which P_1 is idle in σ_o. Moreover $S(T_k, \sigma_c) \leq I$.

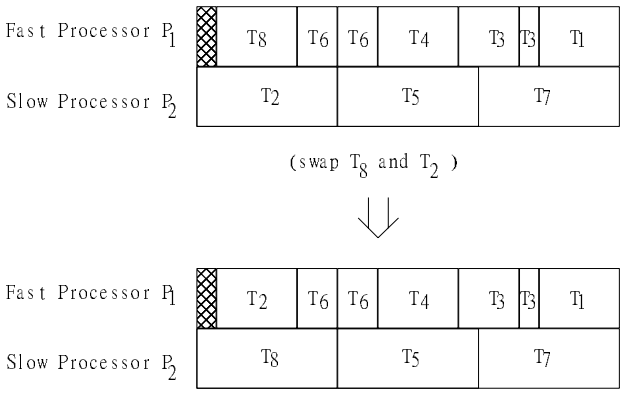

Figure 4. The result of task swapping process

Theorem 1: Given an optimal schedule σ_o with mean flow time F_o, we can find an asymptotically optimal schedule σ_a with mean flow time F_a such that

$$F_a \leq F_o + \frac{c}{n}$$

for some constant c, that is, $\lim_{n\to\infty} F_a = F_o$.

Proof: According to Lemma 1, we know that σ_o can be transformed into a compact schedule σ_c without changing the number of tasks assigned to P_1. Now, we describe a method for constructing an asymptotically optimal schedule σ_a from a compact schedule. The method works as follows: We delay the starting times of all tasks on P_1 in σ_c so that b is the earliest cross point in the resultant schedule

schedule σ_c'. We note that either both processors complete executing their last tasks at the same time, or P_2 becomes idle first. Without loss of generality, we assume the latter. Let T_j be the first task on P_1 that is executed after processor P_2 completes its last task T_n and t_1 denotes the starting time of T_j. In other words, $t_1 = S(T_j) \geq C(T_n)$. Now, we remove all tasks on both processors in the time interval $[0,b]$ temporarily and move all tasks in the time interval $[b, t_1]$ forward to the time interval $[0, t_1 - b]$. After this movement, no task is assigned in the time interval $[t_1 - b, t_1]$. We then assign all temporarily removed tasks (they are scheduled in the time interval $[0,b]$ originally) to this interval. However, rather than in their original order as in σ_c, we reverse the order in which they are assigned. In other words, if T_x is assigned earlier than T_y in the compact schedule σ_c, T_y is assigned earlier than T_x in the new schedule. The segment of σ_c' after t_1 remains unchanged. This new assignment of tasks on processors is illustrated in Figure 5.

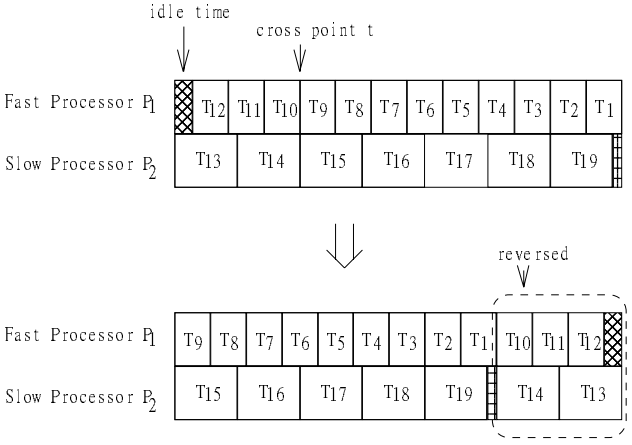

Figure 5. The construction of the asymptotically optimal schedule

When all tasks are scheduled in this manner, we have a new schedule, denoted by σ_a. The mean flow time of σ_a can be computed as follows:

Let W_o be the total flow time of σ_o (e.q., $W_o = F_o \times n$). Let W_{o1} and W_{o2} be the total flow time of the tasks executed on processor P_1 and P_2, respectively. The total flow time W_a of the new schedule σ_a can also be partitioned into two parts, W_{a1} and W_{a2}, where W_{a1} and W_{a2} are the total flow time of the tasks assigned to P_1 and P_2 in σ_a, respectively. We have the following equations:

$$W_o = W_{o1} + W_{o2}$$

$$W_a = W_{a1} + W_{a2}$$

Let us consider the total flow time of the tasks on processor P_1 first. P_1 may be idle for some times in the optimal schedule σ_o. But in schedule σ_a, the idle times on P_1 are late than in σ_o. Hence W_{a1} is less than or equal to the total flow time of all tasks on P_1 in σ_a. That is, $W_{a1} \leq W_{o1}$.

Next, we consider the total flow time of the tasks on P_2. In σ_a, P_2 idles at most once. Let c_1 denote the number of tasks on P_2 scheduled after the idle interval, then c_1 is a constant. Furthermore, the length of the idle interval, denoted by c_2, is less than 1. We have the following equations.

$$W_{a2} \leq \sum_{i=1}^{n-k-c_1} i + \sum_{i=n-k-c_1+c_2}^{n-k+c_2} i = \sum_{i=1}^{n-k} i + c_1 \times c_2$$

but

Let $c = c_1 \times c_2$, then

$$W_{a2} - W_{o2} \leq c$$

and

$$F_a - F_o \leq \frac{c}{n}$$

Therefore, $F_a = F_o$ when the number of tasks approaches ∞.

3. THE ALGORITHM OF FINDING THE ASYMPTOTICALLY OPTIMAL SCHEDULE

According to theorem 1, the asymptotically optimal schedule can be found by the following steps:

(1) Assume that the number of tasks assigned to P_1 is k and find a compact schedule σ_c with k tasks assigned to P_1.

(2) Rearrange the assignment of the tasks in σ_c as described in theorem 1 to produce an asymptotically optimal schedule σ_a.

To do the step (2), we need to find t_1 and t_2 first. We then shift some tasks to the proper time intervals. It is easy to show that the run time for step (2) is linear. To find the time complexity of step (1), we note that the number of tasks assigned to P_1 in σ_o is k, the possible values of k are 0,1,...,n. We try each possible value of k exactly once in order to find out what the exact value of k is.

Lemma 2: Given a fixed k and n tasks $T_1,T_2,...,T_n$ with $R(T_1) \geq R(T_2) \geq ... \geq R(T_n)$. We can find the compact schedule σ_c with k tasks assigned to P_1 in linear time.

Proof: We assign $T_1,...,T_k$ to P_1 and assign T_{k+1}, ...,T_n to P_2. P_1 executes T_k, then executes T_{k-1}, etc.. P_2 executes T_{k+1}, then T_{k+2}, etc.. In a compact schedule, P_2 never idles, but P_1 will idle a short period of times at the beginning. We determinate the initial idle time of P_1 as follows: At first, we execute all tasks on P_1 as early as possible without changing the execution order of tasks. Second, We fix the starting time of T_1 and move all idle time intervals

of P_1 as early as possible. The schedule produced by this method is a compact schedule. Because each task is examined at most twice, the total time in finding this schedule is linear.

According to lemma 2, it is easy to show that the following theorem is true.

Theorem 2: The asymptotically optimal schedule can be found in $O(n^2)$

Proof: By applying the algorithm designed in Lemma 2, we can find an asymptotically optimal schedule corresponding to a fixed k in linear time. We set the value of k to be all possible integer between 1 and n. Therefore, we at most need to try n times and the asymptotically optimal schedule will be found. We need to spend linear time in finding an asymptotically optimal schedule corresponding to a fixed k. Therefore, the total time in finding an asymptotically optimal schedule is $O(n^2)$.

4.Conclusions and Future Works

In this paper, we study the problem of scheduling non-preemptive, independent tasks on two uniform processors to minimize mean flow time. The schedule must satisfy the resource constraints. We propose an algorithm to find an asymptotically optimal schedule. In the special case where the speed ratio between the fast and the slow processor is integer, our algorithm can be simplified to find the optimal schedules. In this paper, we focus on the case of two uniform processor systems. The algorithm proposed in this paper can be extended to handle the multiprocessor systems with arbitrary number of processors. In a multiprocessor system with arbitrary number of processors, our algorithm cannot guarantee that an asymptotically optimal schedule can be found, but a schedule with good approximation factor can be constructed.

5.References

[1] Blazewicz,J., Kubiak,W., Rock,H., Szwarcfiter,J. "Minimizing mean flow-time with parallel processors and resource constraints," Acta informatica. 24, 513-524(1987).

[2] Blazewicz, J., Barcelo, J., Kubiak, W., Rock, H. "Scheduling tasks on two processors with deadlines and additional resources," Eur. J. Oper. Res. 26, 364-370 (1986).

[3] Blazewicz, J., Lenstra, J.K., Rinnooy Kan, A.H.G. "Scheduling subject to resource constraints: classification and complexity," Discrete Appl. Math. 5, 11-24 (1983).

[4] Johnson, D.S. "The NP-completeness column; an ongoing guide.," J. Algorithm 4, 189-203 (1983).

[5] Lenstra, J.K., Rinnooy Kan, A.H.G. "Scheduling theory since 1981: an annotated bibliography," In M.O.H. Eighertaigh, J.K. Lenstra, A.H.G. Rinnooy Kan (eds.), Combinatorial Optimization: Annotated Bibliographies. Chichester: Wiley 1985.

[6] Marco Spuri and John Stankovic. "How to Integrate precedence Constraints and Shared Resources in Real-Time Scheduling," IEEE Trans. on Computers. vol. 43. No. 12, Dec. 1994.

[7] Shlomi Dolev and Alexnder Keizelman. "Non-preemptive Real-Time Scheduling of Multimedia Tasks," Real-Time Systems. Vol. 17. No. 1, 1999.

[8] Yueh-Min Huang and Ruey-Maw Chen. "Scheduling Multiprocessor Job with Resource and Timing Constraints Using Neural Networks," IEEE Trans. on Systems, Man, and Cybernetics-Part B: Cybernetics. vol. 29. No. 4, Aug. 1999.