STAN-CS-73-350 SU-SEL-73-009

An Almost-Optimal Algorithm for the Assembly Line Scheduling Problem

by

Marc T. Kaufman

January 1973

Technical Report No. 53

This work was supported in part by the
National Science Foundation under
Grant GK 23315 and by the Joint
Services Electronic Programs U.S. Army,
U.S. Navy and U.S. Air Force under
Contract N-00014-67-A-0112-0044.

DIGITAL SYSTEMS LABORATORY

STANFORD ELECTRONICS LABORATORIES

STANFORD UNIVERSITY • STANFORD, CALIFORNIA

STAN-CS-73-350 SEL 73-009

AN ALMOST-OPTIMAL ALGORITHM FOR THE ASSEMBLY LINE SCHEDULING PROBLEM

by

Marc T. Kaufman

January 1973

Technical Report No. 53

DIGITAL SYSTEMS LABORATORY
Department of Electrical Engineering Department of Computer Science
Stanford University
Stanford, California

This work was supported in part by the National Science Foundation under Grant GK 23315 and by the Joint Services Electronic Programs U.S. Army, U.S. Navy and U.S. Air Force under contract N-00014-67-A-0112-0044.

AN ALMOST-OPTIMAL ALGORITHM FOR THE ASSEMBLY LINE SCHEDULING PROBLEM

ABSTRACT

This paper considers a solution to the multiprocessor scheduling problem for the case where the ordering relation between tasks can be represented as a tree. Assume that we have n identical processors, and a number of tasks to perform. Each task T_i requires an amount of time μ_i to complete, $0 < \mu_1 \leq k$, so that k is an upper bound on task length. Tasks are indivisible, so that a processor once assigned must remain assigned until the task completes (no preemption). Then the "longest path" scheduling method is almost-optimal in the following sense:

Let ω be the total time required to process all of the tasks by the "longest path" algorithm.

Let ω_o be the minimal time in which all of the tasks can be processed.

Let ω_P be the minimal time to process all of the tasks if arbitrary preemption of processors is allowed.

Then: $\omega_P \leq \omega_o \leq \omega \leq \omega_P + k - k/n$, where n is the number of processors available to any of the algorithms.

INDEX TERMS: Multiprocessing, Parallel Processing, Optimal Scheduling, Tree Graphs, Assembly line Problem

TABLE OF CONTENTS

LIST OF FIGURES

1. Introduction

The "assembly line" problem is well known in the area of multiprocessor scheduling. In this problem, we are given a set of tasks to be executed by a system with n identical processors. Each task, T_i, requires a fixed, known time μ_i to execute. Tasks are indivisible, so that at most one processor may be executing a given task at any time; and they are uninterruptible, so that a processor, once assigned a task, may not leave it until the task is complete. The precedence ordering restrictions between tasks may be represented by a tree (or forest of trees) graph. A task may not be started until all of its predecessors have finished.

This paper examines the execution of such a set of tasks using the "longest path" scheduling algorithm. The longest path algorithm assigns free processors at any time to those available tasks which are furthest from the root of the tree. Processors are never left idle if they can be assigned. T.C. Hu investigated this algorithm for the case where all tasks are the same (unit) length [Hu 61]. He showed that the total execution time is minimal. That is, given the same number of processors, no other non-preemptive algorithm will complete the tasks in less total time.

The results of this paper show that the longest path algorithm remains "almost" optimal when arbitrary times are allowed for each task. In particular, the following relations hold:

Let ω be the total time required to process all of the tasks by the longest path algorithm.

Let ω_o be the minimal time in which all of the tasks can be processed by any nonpreemptive algorithm.

Let ω_p be the minimal time to process all of the tasks if arbitrary preemption of processors is allowed.

Then: $w_p \le \omega_o \le \omega \le \omega_p + k - k/n$

where recalling the definitions above, n is the number of processors used by any of the algorithms and k is an upper bound on task length.

Section 2 of this paper gives a labeling procedure which allows one to find the tasks which are furthest from the root at any time. Section 3 develops the theorem for the case where all task times are integers. Section 4 provides examples which show the inequalities to be tight. Section 5 extends the result to tasks with arbitrary execution times. Finally, Section 6 compares this result to other published results for related problems.

2. Task labeling

The following algorithm allows one to label the tasks in a tree graph with their level, or distance to the root of the tree:

1. If task T_i is a root node (has no successors), the level of T_i is μ_i.
2. If T_i is a node whose successor, S, is at level $\ell(S)$, the level of T_i is $\ell(S) + \mu_i$.

Note that higher levels are further from the root.

3. Longest Path Scheduling -- The Discrete Case

Let us consider the assembly line problem for the case where the task lengths (μ_i) are all integers. If T_i is any task, $\mu_i \in \{1,2,3,\ldots,k\}$ only. Graphically, we can represent the tasks with their precedence relations and execution times as shown in Fig. 1.

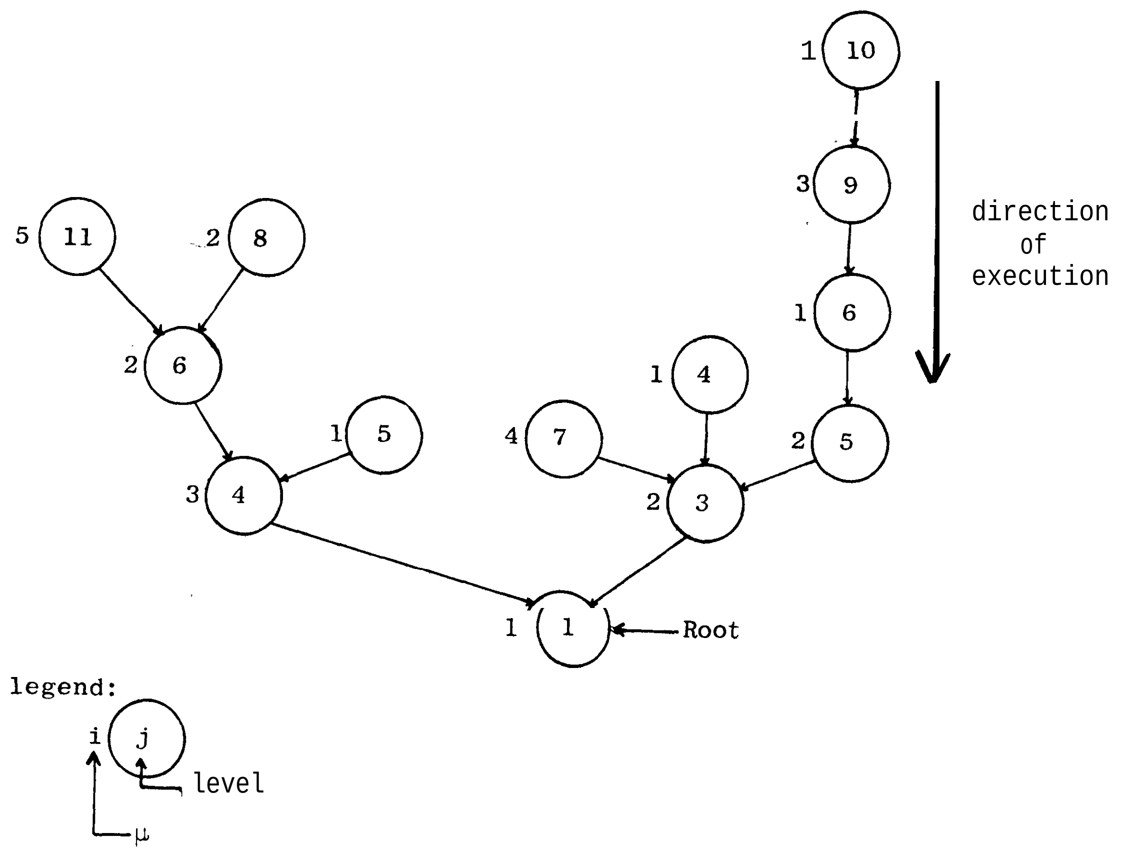


Fig. 1. Example of a Tree of Tasks

We first consider an execution procedure which violates the uninterruptability condition and which allows tasks to be interrupted and processors to be reassigned after each unit of time. For this reason, it is more convenient to represent a task of length m by a chain of m tasks of length 1, as illustrated by Fig. 2. The "double bond" symbol is used to indicate we cannot reassign a processor that is working on such a chain when preemption is not allowed. Since all tasks in the rewritten graph have $\mu=1$, we no longer need to state this explicitly. It is easy to see that each chain-task corresponds to a particular multi-unit task in the original graph. Also, one can quickly verify that the level of the task-head node (the node furthest from the root) of a chain-task is the same as the level of the corresponding multi-unit task.

Now consider the execution of a tree of tasks, T, by two longest path algorithms, G and H. Algorithm G corresponds to the case in which processor preemption during a multi-unit task is not allowed:

Algorithm G: At any time, t, assign the n processors as follows:

(1) If a processor was assigned at time t-l to a task that is connected to its successor by a double bond, assign the processor to that successor (i.e. we are in a multi-unit task, so stay with that task).

(2) Otherwise assign the processor to (one of) the highest level task in T that is ready to be executed. This task will always be a task-head.

Algorithm H is an optimal algorithm (one which gives a minimal total execution time) in which we permit the reassignment of any or all processors

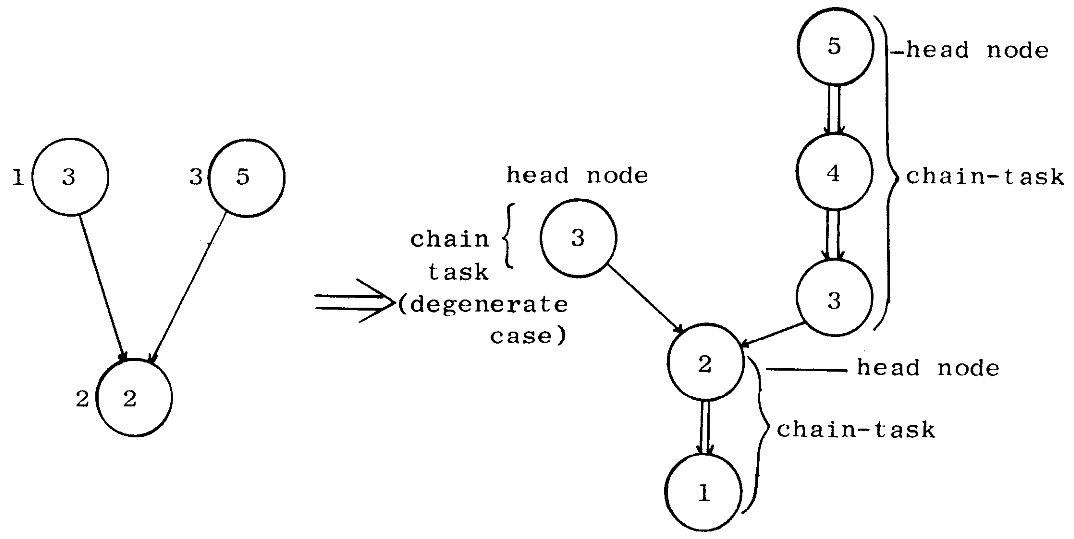


Fig. 2. Rewriting Chains of Tasks

at each unit of time, ignoring the uninterruptability condition. Since all tasks in the rewritten graph are of unit length we may use Hu's algorithm, as it is known to be optimal. The G and H algorithms are identical if there are no multi-unit tasks.

Let ω_p be the time needed to execute the graph by algorithm H. Let ω_0 be the time needed to execute the graph in minimum time, under the restriction that multi-unit tasks not be interrupted. Let ω be the time needed to execute the graph by algorithm G, which also may not interrupt multi-unit tasks. Then:

$$\omega_p \leq \omega_0 \qquad (1)$$

$$\omega_o \leq \omega \qquad (2)$$

Equation (1) follows from the fact that the possible sequences of task assignments by any restricted algorithm are a subset of the possible sequences available to the optimal unrestricted algorithm, H. Equation (2) follows from the definition of an optimal algorithm, since G is also restricted, and so its sequence of task assignments is a possible choice for the algorithm which found ω_o.

As the tree is executed by either algorithm, its depth (the maximum level of any nodes remaining in the tree) decreases with time. Note that the depth cannot decrease faster than one level per unit of time, since it takes (by definition) one time unit to execute a task at any level, and it must be completely finished before its successor can start. We denote the depth of the tree by $d_G(t)$ or $d_H(t)$, accordingly as the algorithm executing the tree is G or H. The following equations derive from the definitions:

$$d_G(0) = d_H(0) = \max_i [\text{level of } T_i]$$

$$d_G(\omega) = d_H(\omega_p) = 0$$

$$d(t) \geq d(t+1) \geq d(t)-1$$

We are interested in a particular time, t', in the execution by algorithm G. This is the earliest time at which the depth of the tree decreases by one at each further step of execution. Specifically:

$$d_G(t)-1 = d_G(t+1) \quad \text{for all } t,\ t' < t < \omega.$$

But, $d_G(t'-1) = d_G(t')$.

We will reach t' no later than time $\omega-1$, since the last unit of execution time must remove one level of the tree (the root level). On the other hand t' may be zero, meaning that the depth of the tree decreases by one <u>every</u> unit of time. If so, G is optimal since no algorithm can go faster, Hence $\omega=\omega_P$ in this case.

For $t' > 0$, the following two lemmas are needed.

Lemma 1: Algorithm G uses all n processors at each time unit up to t' .

Proof:

1. G assigns n processors at each time, until there are fewer than n leaf nodes in the reduced tree. Because we are executing a tree, the number of leaf nodes at each step is nonincreasing with time. Therefore the number of processors assigned at each step is also nonincreasing with time.
2. The highest level in the tree is the same at t'-1 and at t', by the definition of t'. Call this level ℓ. Hence there is some node at level ℓ that was not assigned at t'-1. But this node was available for assignment at t'-1 because ℓ is the highest level in the reduced tree; and the node therefore had no unexecuted predecessors. Thus, there are not enough free processors at time t'-1 to cover all of the unexecuted nodes at level ℓ, so all n processors must be assigned at time t'-1.
3. By (1), all n processors are assigned at all earlier times also. <u>QED.</u>

Corollary 1: At time t' (i.e. after t' units of time have elapsed), algorithm G has completed execution of a total of nt' unit tasks.

Proof: Immediate.

Lemma 2: At time t', suppose algorithm G reduces the highest level in the tree to $d_G(t')$. Then the lowest level at which algorithm G has executed any unit tasks (nodes) is at or above $d_G(t')-(k-1)$.

Proof:

We show that, at t', G has not completed execution of any task-head nodes at a level less than $d_G(t')$. Then since no multi-unit task is longer than k units, the lowest level unit-task in a task-chain which has been executed can be no lower than level $d_G(t')-(k-1)$.

Assume (by way of contradiction) that a task-head node of level less than $d_G(t')$ was assigned at a time $t < t'$. Then there must have been, at this time t, fewer than n leaf nodes at levels at or above $d_G(t')$. Since nodes have at most one successor, execution of those nodes could not leave more than n leaf nodes at or above level $d_G(t')$ at any time after time t. Since all of the leaf nodes which are at or above level $d_G(t')$ at time t are actually assigned to processors at t, we can reduce the depth of the tree by one level at each subsequent time unit, down through level $d_G(t')$. Then, by the definition of t', we can remove one level at each subsequent time unit. So t is an earlier time for which algorithm G begins removing one level of the tree at each time unit. But t' is the earliest time for which this property holds (contradiction). So there is no assignment to such a task-head at time $t < t'$.

<u>QED</u>.

Theorem: $\omega \leq \omega_P + k - \lceil k/n \rceil$

Proof:

At t', G has executed nt' nodes (by Corollary 1).

The 'lowest level executed by G at time t' is at or above level $d_G(t')-(k-1)$ (by Lemma 2). Let ℓ be the lowest level at which any node has been executed by G. Let $q = d_G(t')-\ell+1$. It requires q more units of time to complete all unit tasks in the tree that are at levels at or above level ℓ (since we <u>will</u> complete one level per unit of time from now on), This means that there are at least $(nt'+q)$ unit tasks at or above level ℓ, in total. G then requires $\ell-1$ more units of time to complete the remainder of the tree (i.e. all those nodes at levels less than ℓ).

Algorithm II must execute at least $(nt'+q)$ nodes to complete all nodes in the tree at or above level ℓ, after which it also has $\ell-1$ more levels to execute to complete the tree. So:

for G: $\omega = t' + q + \ell-1$

for H: $\omega_p \geq \left\lceil \dfrac{nt' + q}{n} \right\rceil + \ell-1$

since it takes at least $\left\lceil \dfrac{nt' + q}{n} \right\rceil$ units of time to finish $(nt' + q)$ nodes with n processors.

Then:

$$\omega - \omega_p \leq (t' + q + \ell-1) - \left(\left\lceil \frac{nt' + q}{n} \right\rceil + \ell-1 \right)$$

$$\leq t' + q - \lceil t' + q/n \rceil$$

$$\leq t' + q - t' - \lceil q/n \rceil$$

$$\leq q - \lceil q/n \rceil$$

Now, since

$$q = d_G(t') - \ell + 1$$

and $\ell > d_G(t') - (k-1)$

we have:

$$q \leq d_G(t') - (d_G(t') - (k-1)) + 1$$

SO, $q \leq k$, and we can write:

$$\omega \leq \omega_P + k - \left\lceil \frac{k}{n} \right\rceil \qquad (3)$$ <u>QED</u>.

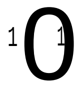
Then, combining (1), (2) and (3):

$$\omega_p \leq \omega_o \leq \omega \leq \omega_P + k - \lceil k/n \rceil . \qquad (4)$$

4. Examples

The following; constructions demonstrate that each combination of inequalities is attainable, thus Proving (4) to be a tight: bound.

Example 1: $\omega_p = \omega_o = \omega$

Trivially, any- tree with unit tasks only, e.g. 

Example 3: $\omega_o = \omega = \omega_p + k - \lceil k/n \rceil$

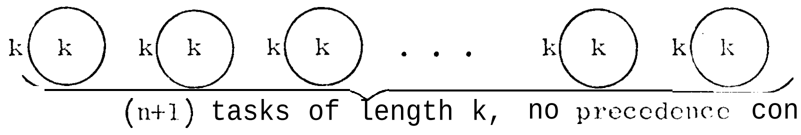


Figure 3. A system that illustrates $\omega_o = \omega = \omega_p + k - \lceil k/n \rceil$

It takes 2k units of time to complete these n+1 tasks by algorithm G, H can complete them in $\lceil k(n+1)/n \rceil$ units of time.

$$2k - \lceil k(n+1)/n \rceil = k - \lceil k/n \rceil$$

Example 3: $\omega_p = \omega_o$, $\omega = \omega_p + k - \lceil k/n \rceil$

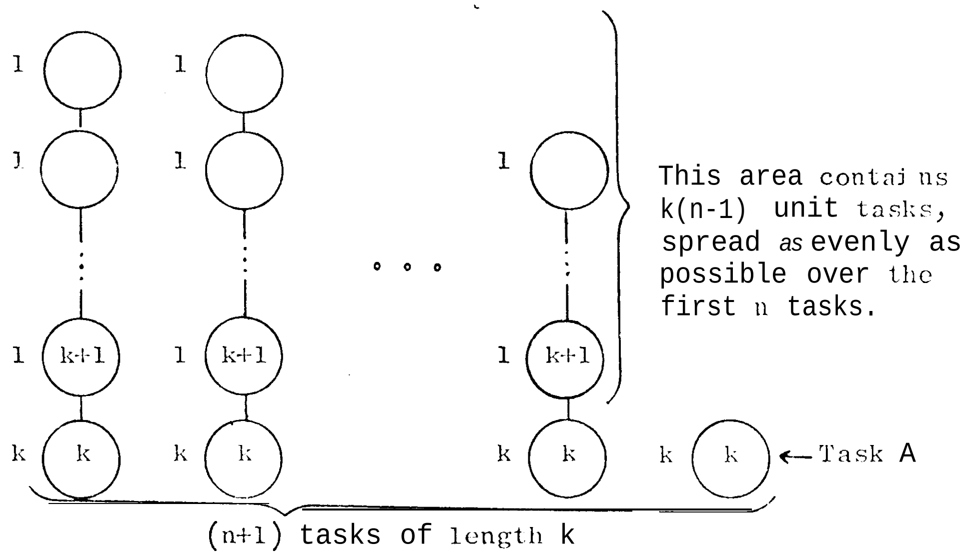

1
1
1
1
1
1
1
k+1
1
k+1
1
k+1
k
k
k
k
k
k
k
k
o o o
This area contains k(n-1) unit tasks, spread as evenly as possible over the first n tasks.
Task A
(n+1) tasks of length k

Figure 4. A system that illustrates $\omega_p = \omega_o$, $\omega = \omega_p + k - \lceil k/n \rceil$

The optimal completion time for this system is reached by starting one processor on task A (of length k) and the other n-l processors on the unit tasks. The optimal time is 2k. Algorithm G completes all of the tasks above the length k tasks and then one of the length k tasks before it starts task A . Algorithm G's time is:

$$2k + \left\lfloor \frac{k(n-1)}{n} \right\rfloor .$$

The difference between these two times is:

$$\left\lfloor \frac{k(n-1)}{n} \right\rfloor = \left\lfloor k - \frac{k}{n} \right\rfloor = k - \left\lceil \frac{k}{n} \right\rceil .$$

5. Extension to noninteger Tasks

This result can be extended to trees for which tasks take arbitrary time, so long as the times are all mutually commensurable.

Let e be the largest real number such that all task times are multiples of e. Muntz and Coffman have shown [Mu70] that there is an integer, s, such that if each task is split into chain-tasks, with each node of length e/ns, application of Hu's algorithm to the resulting graph yields an execution time which is minimal among all algorithms, including processor sharing and arbitrary preemption. Application of algorithm G, of course, yields the same results as before because the chain tasks cannot be broken.

This extension is reducible to the integer case, where the basic unit of time is e/ns rather than 1. So we can rewrite (4) as:

$$\omega_p \le \omega_o \le \omega \le \omega_p + (e/ns)\left(\frac{k}{e/ns} - \left\lceil \frac{k}{n(e/ns)} \right\rceil\right)$$

$$\omega_P \le \omega_O \le \omega \le \omega_P + k - (e/ns)\left\lceil \frac{ks}{e} \right\rceil$$

However, since e divides k, $\frac{ks}{e}$ is integral and

$$\omega_P \le \omega_O \le \omega \le \omega_P + k - (e/ns)(ks/e)$$

$$\omega_P \le \omega_O \le \omega \le \omega_P + k - (k/n). \qquad (5)$$

6. Comparison With Other Results

Hu's algorithm is optimal for trees with unit tasks only. Manacher [Ma 67] and Graham [Gr 66, Gr 69, Gr 72] have investigated longest-path scheduling for structures other than trees. Manacher, using simulation, observed that longest-path schedules tended to be close (within 15%) to optimal in a small set of test cases. Graham has shown in [Gr 66] that, for general directed acyclic graphs, the ratio between the time required to execute the graph with a random list and the optimal time is given by:

$$\omega/\omega_0 \leq 2 - 1/n$$

He also conjectured that this ratio can be improved if a longest-path schedule is used, to:

$$\omega/\omega_0 \leq 2 - 2/(n+1)$$

though this has not yet been proved.

In [Gr 69, Gr 72] Graham presents bounds on execution time for systems in which the tasks are all independent (no precedence constraints). He showed that the "decreasing list" schedule, which is equivalent to a longest-path schedule in this case, satisfies:

$$\omega/\omega_0 \leq 4/3 - 1/3n$$

Since this represents a line with a crossing at the origin, it is better than the result of this paper for small values of ω_0. However, the slope of this line is greater than unity while the slope of the inequality in (5) is exactly unity. For $\omega_0 > 3k$, (5) is a better bound.

Again in [Gr 72] Graham noted (without proof) that, for independent tasks:

$$\omega/\omega_o \leq 1 + n\beta \qquad (6)$$

where:

$$\beta \geq \max \mu(T) / \sum_T \mu(T)$$

In the terminology of this paper, $\max_T \mu(T)=k$, the length of the longest task, and:

$$\sum_T \mu(T) \leq n\omega_p$$

If we then approximate β by $k/n\omega_p$, we can rewrite (6) as:

$$\omega/\omega_o \leq 1 + \frac{k}{n\omega_p} \cdot n$$

However, dividing both sides of (5) by ω_o gives us the slightly better bound:

$$\omega/\omega_o \leq \frac{\omega_p}{\omega_o}\left(1 + \frac{k}{n\omega_p} \cdot (n-1)\right)$$

7. Conclusions

In this paper we have considered the "longest-path" scheduling algorithm as an "almost" optimal algorithm for the scheduling of trees. An upper bound on the execution time for this algorithm is presented. and shown to be better than previous upper bounds for related problems.

BIBLIOGRAPHY

[Gr 66] Graham, R.L., "Bounds for Certain Multiprocessing Anomalies," Bell System Tech. J., Vol. 45, No. 9, Sept. 1966, pp. 1563-1581.

[Gr 69] ____________, "Bounds on Multiprocessing Timing Anomalies," SIAM J. Appl. Math, Vol. 17, No. 2, March 1969, pp. 416-429.

[Gr 72] ____________, "Bounds on Multiprocessing Anomalies and Packing Algorithms," Proceedings SJCC, Vol. 40, 1972, pp. 205-217.

[Hu 61] Hu, T.C., "Parallel Sequencing and Assembly Line Problems," Operations Research, Vol. 9, No. 6, Nov. 1961, pp. 841-848.

[Ma 67] Manacher, G.K., "Production and Stabilization of Real-Time Task Schedules," J. ACM, Vol. 14, No. 3, July 1967, pp. 439-465.

[Mu 70] Muntz, R.R., and E.G. Coffman, Jr., "Preemptive Scheduling of Real-Time Tasks on Multiprocessor Systems," J. ACM, Vol. 17, No. 2, April 1970, pp. 324-338.