Allocation algorithms for dynamically microprogrammable multiprocessor systems

D. M. Nessett* and O. W. Rechard†

Due to advances in integrated circuit technology, multiprocessor systems with more than two Central Processing Units are becoming economically feasible. For this reason, the study of such systems with the aim of developing more efficient hardware and software structures is gaining interest. One type of multiprocessor structure which has recently received some attention is the dynamically microprogrammable multiprocessor system. This paper investigates the problems of efficiently allocating the processors of such a system among the system's workload.

A collection of algorithms and scheduling procedures is developed which may be used for processor allocation in dynamically microprogrammable multiprocessor systems with an arbitrary number of processors. Proofs that these algorithms produce configurations which optimise certain measures are given. The algorithms are practical in the sense that the computation required increases linearly with respect to the number of processors to be allocated. Finally, a simulation study has shown that in many circumstances the use of these algorithms can be expected to yield a significant improvement in system performance over an allocation scheme previously proposed.

(Received August 1977)

1. Introduction

With the significant decrease in hardware costs due mainly to improved techniques in integrated circuit technology, it is becoming economically feasible to design and build large multiprocessor systems (i.e. computer systems with four, six, eight or more processors). While such systems are not widely available today, it is apparent that they will be in the future. If these systems are to be used efficiently, methods of allocating the set of processors among the workload must be developed.

This paper will present a set of algorithms and decision procedures for multiprocessor scheduling. We are not proposing another solution of the general allocation problem as studied by several authors (Arthanari and Mukhopadhyay, 1971; Bauer, 1972; Bauer and Stone, 1970; Coffman and Graham, 1972; Hu, 1961; Szwarc, 1968). Our purpose is rather to develop a set of easily applied algorithms that will allow us to take into account time delays involved in reconfiguring dynamically microprogrammable processors as they are shifted from one set of tasks to another. We assume no prior knowledge (except in a statistical sense) of task execution times or task ordering.

We will consider a multiprocessor system consisting of a large number of identical processors each of which is capable of being dynamically microprogrammed (such a system has been recently proposed by the Burroughs Corp. (Davis, Zucker and Campbell, 1972)). Each job in the system will consist of a set of tasks. Each task, when it is completed, will pass control to the next task or tasks in the job's schedule. The job will terminate when all of its component tasks have been completed. We will assume that microprograms have been developed which are tailored for particular types of tasks. For example, one microprogram might emulate a stack machine and be used for all compiling tasks (e.g. ALGOL, COBOL, FORTRAN, etc. compilations). Another microprogram could be designed for I/O processing (channels would be unnecessary in this type of system, since properly microprogrammed processors could perform all channel functions). Thus, we can classify each task in the system by the microprogram under which it runs. All tasks which run under the same microprogram will be said to belong to the same task class. All tasks in a particular task class will be queued for service in one queue and each task class will be assigned a set of processors (all task classes with non-empty queues will receive at least one processor).

We now describe the basic scheduling philosophy to be used. Assume that statistics for each task class have been gathered which allow us to guess the service time needed for a single processor to process a particular task class. Call this estimated service time P_i (for the i^{th} task class). Use the set of P_is to configure the system (i.e. assign processors to task classes and microprogram the processors for their task classes). After an interval of time I has passed, re-examine the system (i.e. recompute the P_is based on the new system state) and reconfigure the system. This scheme can be thought of as taking a snapshot at time t and acting upon the information gained, then taking another snapshot at time $t + I$ and acting upon the new information. Since smaller values of I will normally be associated with higher system overhead, an interval should be found which is as large as possible but does not degrade system performance by allowing processors to become idle. By the selection of an appropriate value of I and configuring through the use of a good configuration algorithm, it should be possible to keep all processors busy.

2. The basic algorithms

Two basic allocation schemes will be presented. These schemes are named minimax and minisum allocation. In minimax allocation we try to 'balance' the allocation of the processors among the task classes by minimising $\max\left\{\frac{P_i}{N_i}\right\}$; where N_i is the number of processors assigned to task class i and P_i is an estimate of the service time required for one processor to process all tasks currently in the queue. This minimisation is performed over all possible assignments $\bar{N} = (N_1, \ldots, N_L)$ of the K processors among the L tasks. It is evident that $\max\left\{\frac{P_i}{N_i}\right\}$ is approximately the time it will take to finish servicing all tasks presently queued in the system.

Minisum allocation has a somewhat different purpose than minimax. It attempts to use probabilistic information to allow the lengthening of I (while maintaining a good level of system efficiency) and thus lowering the overhead involved in reconfiguration. We first assume that the interval I is greater than $\max\{P_i\}$. Let R_i be the ratio of total service time required

*Lawrence Livermore Laboratory, Livermore, California 94550, USA
†Mathematics Department, University of Denver, University Park, Denver, Colorado 80208, USA

by all of the new tasks entering task class queue i to the total service time of new tasks entering the system over an arbitrary interval of time. We will also assume that this ratio is constant and stable. Defining the idle time of task class i as $I - \frac{P_i}{N_i}$, we will attempt to increase the idle time of task classes with high R_i and decrease the idle time of task classes with low R_i. This will tend to increase the number of processors assigned to those task classes which we expect will receive a high proportion of the new service time requirements and decrease the number of processors assigned to those task classes which we expect will receive a low proportion of the new service time requirements. We can accomplish this manipulation of the idle time of the task classes by maximising the total weighted idle time $\sum \left(I - \frac{P_i R_i}{N_i}\right) = L \cdot I - \sum \frac{P_i R_i}{N_i}$, over all possible configurations $\bar{N}$. Since we have assumed $I > \max \{P_i\}$, we will maximise this sum by minimising $\sum \frac{P_i R_i}{N_i}$.

In practice the interval of reconfiguration I may be shorter than $\max \{P_i\}$. It should be understood that nothing is lost by this being so except increased overhead. To assume I is large and to allocate accordingly is to assume the system is predictable (i.e. that R_i is a constant). If I is in fact smaller than assumed, it is because this erroneous assumption is being corrected. Defining P_i^* to be $P_i \cdot R_i$, we will develop algorithms which minimise $\max \left\{\frac{P_i}{N_i}\right\}$ and $\sum \left\{\frac{P_i^*}{N_i}\right\}$ over all possible configurations $\bar{N} = (N_1, \ldots, N_L)$, $\Sigma N_i = K$, N_i positive integer. (From this point we will use T_i to represent both P_i and P_i^*. We do this for notational simplicity.) Two algorithms will be presented for each of the two problems. The first algorithms (algorithms A and D) are intended for use when the number of processors is not much greater than the number of task classes. The second algorithms (algorithms C and D23) become computationally efficient when the number of processors is considerably greater than the number of task classes.

We begin the solution of the minimax and minisum problems by solving their real counterparts. That is, we allow N_i to assume any real value in the interval $(0, K)$ requiring $\sum N_i = K$. It is simply demonstrated that

$$N_i = RMM(i) \equiv \left[\frac{T_i}{\Sigma T_j} \cdot K\right]$$

will yield the smallest $\max \left\{\frac{T_i}{N_i}\right\}$, for all possible real configurations $\bar{N}$. Moreover, for this choice of N_i all components of the vector $\left(\frac{T_i}{N_i}\right)$ will be equal to $\frac{\Sigma T_j}{K}$.

Through the use of Lagrange multipliers it can be shown that $N_i = RMS(i) \equiv \left[\frac{\sqrt{T_i}}{\Sigma \sqrt{T_j}} \cdot K\right]$ will yield the smallest $\sum \frac{T_i}{N_i}$, for all possible real configurations $\bar{N}$.

At this point it is expedient to introduce some notation: Vectors will be written as a barred letter. Vector components will be written as the unbarred letter with subscript. For example, $\bar{N} = (N_1, N_2, \ldots, N_L)$. We further introduce the following definitions:

$$f_T(\bar{N}) = \left(\frac{T_1}{N_1}, \frac{T_2}{N_2}, \ldots, \frac{T_L}{N_L}\right).$$

$$\|f_T(\bar{N})\|_\infty = \max \left\{\frac{T_i}{N_i}\right\}, \text{ for all } i.$$

$IMM(i) = [RMM(i)]$, where [ ] is the greatest integer function

$IMS(i) = [RMS(i)]$.

1. *The integer minimax problem*

We will assume that our multiprocessor system and associated jobstream have the following characteristics:

1. No task requires more than one processor.
2. The number of task classes is less than or equal to the number of processors.

These assumptions are necessary for the optimality of the algorithms presented in this section. Section 3 is concerned with generalisations of the basic algorithms which can be used in circumstances when these assumptions are not acceptable. Algorithms A and C will now be presented.

Algorithm A

```
1. $A_i := 1$, for all $i$.
2. ORD($i$) := $T_i/A_i$, for all $i$.
3. If $\Sigma A_i = K$, stop; otherwise continue.
4. Find $h$ such that ORD($h$) = max{ORD($i$)}, for all $i$.
5. $A_h := A_h + 1$; ORD($h$) := $T_h/A_h$; Go to 3.
```

In the proofs that follow, $\bar{A}^i$ will denote the value of the vector $\bar{A}$ immediately before the i^{th} iteration; A_j^i will denote the value of the j^{th} component of $\bar{A}^i$; and unless otherwise noted $\bar{A}$ will denote the value of $(A_1, \ldots, A_L)$ after algorithm A terminates. On the i^{th} iteration, $T_h/A_h^i = \|f_T(\bar{A}^i)\|_\infty$ is reduced to $T_h/(A_h^i + 1)$. This means that $\|f_T(\bar{A}^i)\|_\infty \geq \|f_T(\bar{A}^{i+1})\|_\infty$ and consequently,

$$\|f_T(\bar{A})\|_\infty \leq \|f_T(\bar{A}^j)\|_\infty \leq \|f_T(\bar{A}^i)\|_\infty, i \leq j \quad (2.1)$$

Theorem 2.1.1
A is a maximax solution.

Proof:
Let $\bar{S}$ be a minimax solution and assume $\bar{A}$ is not. Then $\|f_T(\bar{S})\|_\infty < \|f_T(\bar{A})\|_\infty$. On the last iteration of algorithm A there will be some subscript b for which $S_b < A_b$, since otherwise the requirement that $\Sigma S_j = \Sigma A_j = K$ would imply $A_j = S_j$ for all j.

Let i be the last iteration such that $S_j \geq A_j^i$ for all j. Such an iteration exists since $A_j^1 = 1$ and $S_j \geq 1$ for all j. Let

$$T_c/A_c^i = \|f_T(\bar{A}^i)\|_\infty \ .$$

Then

$$A_j^{i+1} = A_j^i, j \neq c$$

and

$$A_c^{i+1} = A_c^i + 1 \ .$$

Since this is the last iteration where $S_j \geq A_j^i$ for all j and since only A_c^i changes, $A_c^i = S_c$. Since A_c^i was chosen for incrementation, $T_c/S_c = T_c/A_c^i \geq T_j/A_j^i$ for all j. But by (2.1)

$$\|f_T(\bar{A})\|_\infty \leq \|f_T(\bar{A}^i)\|_\infty = T_c/A_c^i = T_c/S_c \leq \|f_T(\bar{S})\|_\infty$$

which is a contradiction.

Lemma 2.1.2
$A_i \leq IMM(i) + 1$, for all i.

Proof: As stated previously $\|f_T(\bar{Q})\|_\infty \geq \frac{\Sigma T_j}{K}$ for all possible configurations $\bar{Q}$ of K processors. Thus, $\|f_T(\bar{A})\|_\infty \geq \frac{\Sigma T_j}{K}$.

By (2.1) $\|f_T(\bar{A}^i)\|_\infty \geq \|f_T(\bar{A})\|_\infty \geq \frac{\Sigma T_j}{K}$. Therefore, on each iteration i of algorithm A, there exists h such that $\frac{T_h}{A_h^i} \geq \frac{\Sigma T_j}{K}$.

But $\frac{T_n}{IMM(n) + 1} \geq \frac{T_n}{RMM(n)} = \frac{\Sigma T_j}{K}$ for all n. Consequently, if $A_j^i = IMM(j) + 1$, A_j^i cannot be selected for incrementation.

We now present algorithm C which, as has been previously stated, requires fewer iterations than algorithm A when the number of processors is considerably greater than the number of task classes.

Algorithm C

```
1. $C_i := IMM(i) + 1$, for all $i$.
2. $ORD(i) := T_i/IMM(i)$, for all $i$.
3. Find $h$ such that $ORD(h) = \min\{ORD(i)\}$, for all $i$.
4. $C_h := C_h - 1$; $ORD(h) := T_h/(C_h - 1)$.
5. If $\Sigma C_i = K$, stop; otherwise Go to 3.
```

Theorem 2.1.3
Algorithm C generates a minimax configuration.

Proof:
Assume algorithm C does not generate a minimax configuration. Let $\bar{S}$ be a minimax configuration. Then

$$\|f_T(\bar{S})\|_\infty < \|f_T(\bar{C})\|_\infty .$$

Since

$$\frac{T_i}{IMM(i)+1} < \frac{T_i}{RMM(i)} \leq \|f_T(\bar{S})\|_\infty ,$$

$$\|f_T(\bar{S})\|_\infty > \|f_T(\bar{C}^1)\|_\infty .$$

Let j be the last iteration such that

$$\|f_T(\bar{S})\|_\infty \geq \|f_T(\bar{C}^j)\|_\infty .$$

Let $\frac{T_a}{C_a^j}$ be chosen for decrementation. Then $C_i^{j+1} = C_i^j$, $i \neq a$ and $C_a^{j+1} = C_a^j - 1$. Moreover, $\frac{T_a}{C_a^j - 1} = \min\left\{\frac{T_i}{C_j^i - 1}\right\}$ for all i. By choice $\|f_T(\bar{S})\|_\infty < \|f_T(\bar{C}^{j+1})\|_\infty$. But since C_a^{j+1} is the only component that has changed,

$$\|f_T(\bar{C}^{j+1})\|_\infty = \frac{T_a}{C_a^j - 1} .$$

This implies

$$\frac{T_i}{C_i^j - 1} \geq \frac{T_a}{C_a^j - 1} > \|f_T(\bar{S})\|_\infty \geq \frac{T_i}{S_i}$$

for all i. But this means that $C_i^j - 1 < S_i$ or $C_i^j \leq S_i$ for all i; which in turn implies that $\Sigma C_i^j \leq \Sigma S_i$. But there exists at least a $j + 1$st iteration so that $K < \Sigma C_i^j \leq \Sigma S_i = K$, which is a contradiction.

2. *The integer minisum problem*

We will now turn our attention to the minisum problem.

Algorithm D

```
1. $D_i := 1$, for all $i$.
2. $ORD(i) := \frac{T_i}{2}$, for all $i$.
3. If $\Sigma D_i = K$, stop; otherwise continue.
4. Find $h$ such that $ORD(h) = \max\{ORD(i)\}$, for all $i$.
5. $D_h := D_h + 1$; $ORD(h) := \frac{T_h}{D_h(D_h + 1)}$; Go to 3.
```

In what follows, we will use the fact that if A and B are greater than zero and $A \geq B$, then

$$\frac{1}{A} - \frac{1}{A+1} \leq \frac{1}{B} - \frac{1}{B+1} \qquad (2.2)$$

Theorem 2.2.1
Algorithm D generates a minisum configuration.

Proof:
Algorithm D obviously works for L processors. Therefore, assume it works for $L + j$ processors. Since there are j iterations D_i^{j+1} will be the final configuration generated by algorithm D. Thus,

$$\sum \frac{T_i}{D_i^{j+1}} \leq \sum \frac{T_i}{S_i} ,$$

for all configurations $\bar{S}$ of $L + j$ processors. Let $\bar{Q}$ be any configuration of $L + j + 1$ processors. Then there exists an h such that $Q_h > D_h^{j+1}$ or $Q_h - 1 \geq D_h^{j+1}$. Choose a such that

$$\frac{T_a}{D_a^{j+1}} - \frac{T_a}{D_a^{j+1}+1} = \max\left\{\frac{T_i}{D_i^{j+1}} - \frac{T_i}{D_i^{j+1}+1}\right\} .$$

By (2.2),

$$\frac{T_h}{Q_h - 1} - \frac{T_h}{Q_h} \leq \frac{T_h}{D_h^{j+1}} - \frac{T_h}{D_h^{j+1}+1} \leq \frac{T_a}{D_a^{j+1}} - \frac{T_a}{D_a^{j+1}+1} .$$

Let $\bar{Q}^* = (Q_1, \ldots, Q_h - 1, \ldots, Q_L)$. Then:

$$\sum \frac{T_i}{Q_i^*} - \sum \frac{T_i}{Q_i} = \frac{T_h}{Q_h - 1} - \frac{T_h}{Q_h} \leq \frac{T_a}{D_a^{j+1}} - \frac{T_a}{D_a^{j+1}+1} =$$

$$\sum \frac{T_i}{D_i^{j+1}} - \sum \frac{T_i}{D_i^{j+2}} . \qquad (2.2.1.1)$$

By assumption:

$$\sum \frac{T_i}{D_i^{j+1}} \leq \sum \frac{T_i}{Q_i^*}$$

which means

$$-\sum \frac{T_i}{Q_i^*} \leq -\sum \frac{T_i}{D_i^{j+1}} \qquad (2.2.1.2)$$

Adding equations (2.2.1.1) and (2.2.1.2) we get:

$$-\sum \frac{T_i}{Q_i} \leq \sum \frac{T_i}{D_i^{j+2}}$$

which is equivalent to $\sum \frac{T_i}{Q_i} \geq \sum \frac{T_i}{D_j^{j+2}}$. Since Q was an arbitrary configuration of $L + j + 1$ processors, algorithm D generates a minisum configuration for $L + j + 1$ processors. Consequently, by induction, the theorem is true for all numbers of processors.

We now present the two algorithms which will be used in the construction of algorithm D23.

Algorithm D2

```
1. $D2(i) := IMS(i) + 1$, for all $i$.
2. $ORD(i) := -\frac{T_i}{(D2(i) - 1)\, D2(i)}$, for all $i$.
3. Find $h$ such that $ORD(h) = \max\{ORD(i)\}$, for all $i$.
4. $D2(h) := D2(h) - 1$; $ORD(h) := -\frac{T_h}{(D2(h) - 1)\, D2(h)}$
5. If $\Sigma D2(i) = K$, stop; otherwise Go to 3.
```

Algorithm D3

```
1. $D3(i) :=$ IMS$(i)$, for all $i$.
2. $ORD(i) := \frac{T_i}{(D3(i) + 1)\, D3(i)}$, for all $i$.
3. If $\Sigma D3(i) = K$, stop; otherwise continue.
4. Find $h$ such that $ORD(h) = \max\{ORD(i)\}$, for all $i$.
5. $D3(h) := D3(h) + 1$; $ORD(h) := \frac{T_h}{D3(h)\,(D3(h) + 1)}$;
   Go to 3.
```

Since $RMS(i) = \dfrac{K\sqrt{T_i}}{\Sigma\sqrt{T_i}}, \dfrac{T_i}{RMS(i)^2} = \dfrac{T_j}{RMS(j)^2}$ for all i and j.

From this it can be seen that for all i and j.

$$\frac{T_i}{IMS(i)\,(IMS(i) - 1)} > \frac{T_i}{RMS(i)^2} = \frac{T_j}{RMS(j)^2} > \frac{T_j}{(IMS(j) + 1)\,(IMS(j) + 2)} \qquad (2.3)$$

Lemma 2.2.2
Either $D_i \leq IMS(i) + 1$ for all i or $D_i \geq IMS(i)$ for all i.

Proof:
If there exists a j such that $D_j > IMS(j) + 1$, then by (2,3), all D_i must have been incremented to at least $IMS(i)$ since algorithm D picks out the largest ORD(i). In this case $D_i \geq IMS(i)$ for all i.

Theorem 2.2.3
If $D_i \leq IMS(i) + 1$, algorithm D2 generates an integer minisum configuration.

Proof:
Assume $\bar{D}2 \neq \bar{D}$. Choose h such that $D2_h < D_h$ and choose l such that $D2_l > D_l$. Now consider the configuration $\bar{S}^{(1)}$ where:

$S_h^{(1)} = D_h - 1; S_l^{(1)} = D_l + 1; S_i^{(1)} = D_i, i \neq h, l$.

Since

$$\sum \frac{T_i}{S_i^{(1)}} \geq \sum \frac{T_i}{D_i}$$

then

$$\frac{T_h}{S_h^{(1)}} + \frac{T_l}{S_l^{(1)}} \geq \frac{T_h}{D_h} + \frac{T_l}{D_l}$$

or

$$\frac{T_h}{D_h} - \frac{T_h}{D_h - 1} \leq \frac{T_l}{D_l + 1} - \frac{T_l}{D_l} \qquad (2.2.3.1)$$

But for some iteration j we know that $D2_h^j = D_h$ and $D2_h^j$ is chosen for decrementation. Therefore,

$$\frac{-T_h}{(D_h - 1)\,D_h} \geq \frac{-T_l}{(D2_l^j - 1)\,D2_l^j} .$$

Now $D2_l^j \geq D2_l > D_l$ so we have

$$\frac{T_h}{D_h} - \frac{T_h}{D_h - 1} = \frac{-T_h}{D_h(D_h - 1)} \geq \frac{-T_l}{D_l(D_l + 1)} = \frac{T_l}{D_l + 1} - \frac{T_l}{D_l} . \qquad (2.2.3.2)$$

From (2.2.3.1) and (2.2.3.2) it follows that

$$\frac{T_h}{D_h} - \frac{T_h}{D_h - 1} = \frac{T_l}{D_l + 1} - \frac{T_l}{D_l}$$

or

$$\frac{T_h}{D_h} + \frac{T_l}{D_l} = \frac{T_h}{D_h - 1} + \frac{T_l}{D_l + 1}$$

and consequently,

$$\sum \frac{T_i}{D_i} = \sum \frac{T_i}{S_i^{(1)}} .$$

Thus $\bar{S}^{(1)}$ is a minisum configuration. Moreover we note that $\bar{S}^{(1)}$ is closer to $\bar{D}2$ than is $\bar{D}$ in the sense that

$$\Sigma\,|\,S_i^{(1)} - D2_i| < \Sigma\,|\,D_i - D2_i\,| .$$

If now $\bar{D}2 \neq \bar{S}^{(1)}$, the above argument can be repeated. That is, we can again choose h and l (possibly different) such that $D2_h < S_h^{(1)}$ and $D2_l > S_l^{(1)}$ and define a new configuration $\bar{S}^{(2)}$ where $S_h^{(2)} = S_h^{(1)} - 1$; $S_l^{(2)} = S_l^{(2)} + 1$; $S_i^{(2)} = S_i^{(1)}$, $i \neq h, l$. $S^{(2)}$ will again be a minisum configuration with $\Sigma\,|\,S_i^{(2)} - D2_i\,| < \Sigma\,|\,S_i^{(1)} - D2_i\,|$. Since the sums of absolute values are integers, we will eventually arrive at a minisum configuration $\bar{S}^{(n)} = \bar{D}2$.

An argument parallel to that given above, utilising the fact that algorithm $D3$ always increments values $D3_j$ serves to prove:

Theorem 2.2.4
If $D_i \geq IMS(i)$ for all i, algorithm $D3$ generates an integer minisum configuration.

For a set S let S_i be the ith largest element in S (e.g. $S_1 = \max(S)$, $S_2 = \max(S - \{S_1\}), \ldots$). We define

$$F(x, S) = S_x .$$

*Algorithm D*23
Let $Z = K - \Sigma IMS(i)$

$$\text{If } F\left(Z, \left\{\frac{T_i}{IMS\,(i)\,(IMS\,(i){+}1)}\right\}\right) \geq \max_i \left\{\frac{T_i}{(IMS(i) + 1)\,(IMS(i) + 2)}\right\}$$

then perform algorithm $D2$; otherwise perform algorithm $D3$.

Theorem 2.2.5
Algorithm D23 generates an integer minisum configuration.

Proof:
Let

$$S = \left\{\frac{T_i}{(IMS(i)\,(IMS(i) + 1)}\right\}$$

and

$$R = \left\{\frac{T_i}{(IMS(i) + 1)\,(IMS(i) + 2)}\right\} .$$

We must show that $D_i \leq IMS(i) + 1$ if and only if $F(Z, S) \geq \max_i(R)$.

Let $F(Z, S) \geq \max_i(R)$. By (2.3), no D_i will be incremented to $IMS(i) + 2$ until all D_i are incremented to $IMS(i)$. By assumption:

$$S_1 \geq S_2 \geq \ldots \geq S_Z \geq R_1 \geq \ldots \geq R_L .$$

Thus, Z of the D_i (those corresponding to $S_1, \ldots, S_Z$) will be imcremented to $IMS(i) + 1$ before any D_i are incremented to $IMS(i) + 2$. But $K = Z + \Sigma IMS(i)$, which implies that after Z of the D_i are incremented to $IMS(i) + 1$, all of the processors will have been allocated and hence, no D_i will be incremented to $IMS(i) + 2$. Thus, $F(Z, S) \geq \max_i(R)$ implies $D_i \leq IMS(i) + 1$ for all i.

Now suppose that $D_i \leq IMS(i) + 1$ for all i. Assume $F(Z, S) < \max(R)$. This means that before the D_i corresponding to S_Z is incremented to $IMS(i) + 1$, the D_i corresponding to R_1 will be incremented to $IMS(i) + 2$ which is a contradiction. Thus $D_i \leq IMS(i) + 1$ for all i implies $F(Z, S) \geq \max(R)$. By Lemma 2.4.2, either $D_i \leq IMS(i) + 1$ or $D_i \geq IMS(i)$. Consequently $F(Z, S) < \max(R)$ implies $D_i \geq IMS(i)$ for all i. We can now apply Theorems 2.2.3 and 2.2.4 to complete the proof of Theorem 2.2.5.

3. Generalisations

The preceding section was concerned with the solution of the minimax and minisum problems when two assumptions were made concerning the multiprocessor system and its associated job stream. In this section we will present a set of procedures and algorithms which will operate on multiprocessor systems in which these assumptions are not valid. It is necessary, however, to first indicate the plausibility of multiprocessor systems and job streams which do not conform to these assumptions. The first requirement is that all task classes are able to operate with only one processor. In most cases this does not seem to be an unreasonable assumption. There are possible circumstances, however, when this requirement would not be satisfied. Consider a multiprocessor system similar to the Burroughs system in which it is possible to change the character of a subset of the processors so that one processor in the subset generates hardware control signals which are then sent to all of the other processors in the subset. This capability would allow part of the multiprocessor system to be configured into an array processor. If this capability were present, it would be desirable to allocate processors to an array processing task class in multiples of some minimum number 0_i. This desirability stems from the nature of array processing problems (e.g. matrix computations). Thus, an array processing task class would demand more than one processor before it could begin servicing. Another possible situation in which the first requirement would not be satisfied would be the existence of a task class dedicated to the emulation of a conventional multiprocessor system. The operating system of such an emulated multiprocessor system might expect at least two processors to be present for proper operation. Thus, there may exist task classes in the host multiprocessor system which require a minimum number of processors M_i before they can begin servicing.

The second requirement that the number of task classes be less than or equal to the number of processors is probably unrealistic. For example, this requirement is not satisfied for a 4 processor system with the following task classes:

1. Stack machine processing (e.g. compilations)
2. Arithmetic processing (e.g. numerical computations)
3. An emulator
4. Supervisor and maintenance functions
5. I/O processing.

The next two subsections are concerned with procedures which allow allocation algorithms to be used when the two previous restrictions are violated. The next subsection deals with the removal of restriction 2, while the last subsection deals with the removal of restriction 1.

1. *Removing restriction* 2

Restriction 2 specifies that the number of task classes is assumed to be less than or equal to the number of processors. Upon removal of the restriction, it may occur that there are more task classes contending for service than there are processors to be allocated. In such a case, some of the task classes must be eliminated as candidates for processors. An obvious procedure which would effect this goal would be to order the task classes by their associated T_i and select the highest K for allocation. This assumes restriction 1 is still in force. If this is not the case, select the highest task classes such that the minimum number of processors required is less than or equal to K. Unfortunately, this scheme introduces a new problem. It is possible that by this type of allocation some task classes may be 'frozen out' (i.e. a task class may have a small associated T_i and the other task classes may have a high arrival rate of service time requirements. This situation could result in some of the task classes being denied processors for an indefinite period of time). This inequity can be corrected and priorities among the task classes can be incorporated by the following scheme. Let P_i be the priority of task class i and let C_i be the time since task class i was allocated at least one processor. Let $F(P_i, C_i, T_i)$ be some function such that:

$$F(P_i, C_i, T_i) > F(P_j, C_i, T_i), \text{ for } P_i > P_j.$$

$$F(P_i, C_i^1, T_i) > F(P_i, C_i^2, T_i), \text{ for } C_i^1 > C_i^2 .$$

and

$$F(P_i, C_i, T_i^1) > F(P_i, C_i, T_i^2), \text{ for } T_i^1 > T_i^2 .$$

Also require:

$$\lim_{P_i \to \infty} F(P_i, C_i, T_i) = \infty, \text{ for all } C_i \text{ and } T_i$$

$$\lim_{C_i \to \infty} F(P_i, C_i, T_i) = \infty, \text{ for all } P_i \text{ and } T_i$$

and

$$\lim_{T_i \to \infty} F(P_i, C_i, T_i) = \infty, \text{ for all } P_i \text{ and } C_i$$

An example of such a function is $w_1 P_i + w_2 C_i + w_3 T_i$, where the w_i are constants. In the algorithms of Section 2 replace T_i by $F(P_i, C_i, T_i)$. If the number of task classes exceeds the number of processors, order the task classes by their associated value of F and select the highest K task classes for allocation. This procedure will insure that no task class will ever become 'frozen out'.

2. *Removing restriction* 1

The removal of restriction 1 will require the modification of the previous allocation algorithms. The algorithms presented in this section are generalisations of the algorithms presented in Section 2. The following discussion will employ the notational conventions introduced in Section 2. Five algorithms will be presented. Two are concerned with computing minimax configurations for systems in which the number of processors allocated to task class i is a multiple of some number 0_i. We have been unable to generalise the minisum algorithms of Section 2 for the multiple processors case. Unfortunately, the generalisations of C and D23 do not generate optimum configurations in all possible circumstances. The crux of the problem lies in the fact that it may not be possible to allocate all of the available processors. While this characteristic does no damage to the minimax generalisations, it means for minisum that it is no longer sufficient to allocate processors by examining the consequences of allocating one processor at a time to each task class. The last three algorithms compute minimax and minisum configurations for systems in which the number of processors allocated to task class i must be greater than some minimum number M_i. We have been unable to develop an algorithm for the minimum number of processors problem which would correspond to D23. The difficulty stems from the fact that it is possible that some of the M_i can be less than $IMM(i) + 1$, while some can be greater than $IMM(i)$. Under these circumstances, the number of processors to be allocated may be less than necessary to allocate every task class at least $IMM(i)$ processors. This, of course, destroys the dichotomous case structure necessary in algorithm D23. For the purposes of the following algorithms we will adopt restrictions analogous to restriction 2 above, namely $\Sigma 0_i \leq K$ and $\Sigma M_i \leq K$. The discussion in the preceding section can be adapted to situations in which these restrictions are not satisfied.

Algorithm E

```
1. Mark all $i$ unfinished.
2. $E_i := 0_i$, for all $i$.
3. ORD($i$) $:= T_i/E_i$, for all $i$.
4. Find $h$ such that ORD($h$) $= \max\{T_i/E_i\}$, for all marked
   unfinished.
5. If $\Sigma E_i + 0_h > K$, mark $h$ finished; otherwise Go to 7.
```

```
6. If all $i$ are marked finished, stop; otherwise Go to 4.
7. $E_h := E_h + 0_h$; ORD($h$) $:= T_h/E_h$; Go to 4.
```

By an argument similar to that given for algorithm A

$$\|f_T(\bar{E})\|_\infty \leq \|f_T(\bar{E}^j)\|_\infty \leq \|f_T(\bar{E}^i)\|_\infty;\ i \leq j \qquad (3.1)$$

Lemma 3.2.1
Let $\bar{Q}$ be a minimax configuration under the special constraint $Q_i = m_i \cdot O_i$, m_i a positive integer and assume algorithm E does not generate a minimax configuration under this special constraint. Then on the last iteration p of algorithm E, there exists a subscript k such that $Q_k < E_k^p$.

Proof:
Assume $Q_j \geq E_j^p$ for all j. Since $\bar{E}^p$ is not a minimax configuration under the special constraint, there is a subscript h such that $Q_h > E_h^p$. This implies a positive integer n exists such that $Q_h = E_h^p + n \cdot 0_h$. But algorithm E does not stop unless $\Sigma E_i^p + 0_h > K$. Since $Q_j \geq E_j^p$ for all j,

$$\sum_{j \neq h} Q_j \geq \sum_{j \neq h} E_j^p .$$

This means $\Sigma Q_j \geq \Sigma E_j^p + 0_h > K$ which is a contradiction.

Theorem 3.2.2
Algorithm E generates a minimax configuration under the constraints $E_i = m \cdot 0_i$, m_i a positive integer and $\Sigma S_i \leq K$.

Proof:
Let $\bar{S}$ be a minimax configuration under the special constraint and assume $\bar{E}$ is not. On the last iteration of algorithm E there will be some subscript b for which $S_b < E_b$, by Lemma 3.2.1.

Let i be the last iteration such that $S_j \geq E_j^i$ for all j. Such an iteration exists since $E_j^1 = 0_j$ and $S_j \geq 0_j$ for all j. Let $T_c/E_c^i = \|f_T(\bar{E}^i)\|_\infty$. Then $E_j^{i+1} = E_j^i$ for $j \neq c$ and $E_c^{i+1} = E_c^i + 0_i$. Since this is the last iteration where $S_j \geq E_j^i$ for all j and since only E_c^i changes, $E_c^i = S_c$. Since E_c^i was chosen for incrementation, $T_c/S_c = T_c/E_c^i \geq T_j/E_j^i$ for all j. But by (3.1)

$$\|f_T(\bar{E})\|_\infty \leq \|f_T(\bar{E}^i)\|_\infty = T_c/E_c^i = T_c/S_c \leq \|f_T(\bar{S})\|_\infty$$

which is a contradiction.

Algorithm F

```
1. $F_i := IMM(i) - [IMM(i) \bmod 0_i] + 0_i$, for all $i$.
2. ORD($i$) $:= \dfrac{T_i}{F_i - 0_i}$, for all $i$.
3. Find $h$ such that ORD($h$) $= \min\{$ORD($i$)$\}$, for all $i$.
4. $F_h := F_h - 0_h$; ORD($h$) $:= \dfrac{T_h}{F_h - 0_h}$.
5. If $\Sigma F_i \leq K$, stop; otherwise Go to 3.
```

Theorem 3.2.3
Algorithm F generates a minimax configuration under the constraints $F_i = m \cdot 0_i$, m a positive integer and $\Sigma F_i \leq K$.

Proof:
Assume algorithm F does not generate a minimax configuration. Let $\bar{S}$ be a minimax configuration. Then $\|f_T(\bar{S})\|_\infty < \|f_T(\bar{F})\|_\infty$. Now

$$\|f_T(\bar{S})\|_\infty \geq \frac{T_i}{RMM(i)} = \frac{\Sigma T_i}{K}$$

for all i; since if this were not true,

$$\frac{T_c}{S_c} \equiv \max\left\{\frac{T_i}{S_i}\right\}$$

would imply that for all i

$$\frac{T_i}{S_i} \leq \frac{T_c}{S_c} < \frac{T_i}{RMM(i)} .$$

This would mean $S_i > RMM(i)$ for all i and

$$\Sigma S_i > \Sigma RMM(i) = K$$

which would be a contradiction. Thus,

$$\|f_T(\bar{S})\|_\infty \geq \|f_T(\bar{F}^1)\|_\infty ,$$

since $F_i \geq RMM(i)$ for all i. Let j be the last iteration such that $\|f_T(\bar{S})\|_\infty \geq \|f_T(\bar{F}^j)\|_\infty$. Let F_a^j be chosen for decrementation. Then $F_i^{j+1} = F_i^j$, $i \neq a$ and $F_a^{j+1} = F_a^j - 0_a$. Moreover,

$$\frac{T_a}{F_a^j - 0_a} = \min\left\{\frac{T_i}{F_i^j - 0_i}\right\}$$

for all i. By the choice of j, $\|f_T(\bar{S})\|_\infty < \|f_T(\bar{F}^{j+1})\|_\infty$. But since F_a^{j+1} is the only component that has changed,

$$\|f_T(\bar{F}^{j+1})\|_\infty = \frac{T_a}{F_a^j - 0_a} .$$

This implies:

$$\frac{T_i}{F_i^j - 0_i} \geq \frac{T_a}{F_a^j - 0_a} > \|f_T(\bar{S})\|_\infty \geq \frac{T_i}{S_i}$$

for all i. But this means that $F_i^j - 0_i < S_i$ or $F_i^j \leq S_i$ for all i, which in turn implies that $\Sigma F_i^j \leq \Sigma S_i$. But there exists at least a $j + 1$st iteration so that $K < \Sigma F_i^j \leq \Sigma S_i \leq K$, which is a contradiction.

We will now turn our attention to the case in which a minimum number of processors is required. Three algorithms will be presented. No generalisation of D23 will be given for reasons discussed previously.

Algorithm U

```
1. $U_i := M_i$ for all $i$.
2. ORD($i$) $:= \dfrac{T_i}{M_i}$ for all $i$.
3. If $\Sigma U_i = K$, stop; otherwise, continue.
4. Find $h$ such that ORD($h$) $= \max\{$ORD($i$)$\}$ for all $i$.
5. $U_h := U_h + 1$; ORD($h$) $:= \dfrac{T_h}{U_h}$, Go to 3.
```

By an argument parallel to the proof of Theorem 2.1.2 we have

Theorem 3.2.4.
Algorithm U generates a minimax configuration under the constraint $U_i \geq M_i$ for all i.

Algorithm V

```
1. If $M_i > IMM(i)$, $V_i := M_i$ and mark $i$ finished.
2. If $M_i \leq IMM(i)$ : $V_i := IMM(i) + 1$.
3. ORD($i$) $:= T_i/IMM(i)$, for all $i$ marked unfinished.
4. If $\Sigma V_i = K$, stop; otherwise continue.
5. If all $i$ are marked finished, stop; otherwise continue.
6. Find $h$ such that ORD($h$) $= \min\{$ORD($i$)$\}$, for all $i$ marked
   unfinished.
7. $V_h := V_h - 1$.
8. If $V_h = M_h$, mark $h$ finished, Go to 4.
9. If $V_h > M_h$, ORD($h$) $:= T_h/V_h - 1$; Go to 4.
```

Lemma 3.2.5
If $V_i = M_i$, then $V_i = U_i$,

Proof:
If $M_i \geq IMM(i) + 1$, then since $U_i^1 = M_i$,

$$T_i/U_i^1 \leq T_i/(IMM(i) + 1)$$

But $T_i/(IMM(i) + 1) < T_i/RMM(i) \leq \|f_T(\bar{U})\|_\infty \leq \|f_T(\bar{U}^j)\|_\infty$ for all iterations j. Thus, U_i^1 will never be incremented and $U_i = M_i$. Also since $M_i > IMM(i)$, $V_i = M_i$.

If $M_h \leq IMM(h)$ and $V_h = M_h$, there must exist an iteration j such that $\frac{T_h}{M_h} \leq \frac{T_k}{V_k^j - 1}$ for all indices k where $V_k^j \neq M_k$. By the operation of algorithm V, $V_k^j \geq M_k$ and $\Sigma V_k^j > K$. Since $\Sigma U_k = K$, there exists an index p such that $U_p < V_p^j$. Now $M_p \leq U_p$, so $V_p^j \neq M_p$. Since U_p and V_p^j are integers, $U_p \leq V_p^j - 1$ which means $\frac{T_p}{U_p} \geq \frac{T_p}{V_p^j - 1}$. By the hypothesis of

iteration j, $\frac{T_p}{U_p} \geq \frac{T_h}{M_h}$. But

$$\|f_T(\bar{U}^q)\|_\infty \geq \|f_T(\bar{U})\|_\infty \geq \frac{T_p}{U_p} \geq \frac{T_h}{M_h}$$

for all iterations q. This means that for every iteration of algorithm U, $U_h^1 = M_h$ will imply ORD(h) is never maximum and thus $U_h = M_h$.

Theorem 3.2.6
Algorithm V generates a minimax configuration under the constraint $V_i \geq M_i$ for all i.

Proof:
Assume algorithm V does not generate a minimax configuration under the stated constraint. Then $\|f_T(\bar{U})\|_\infty < \|f_T(\bar{V})\|_\infty$. Since $V_i^1 \geq IMM(i) + 1$ for all i, $\|f_T(\bar{V}^1)\|_\infty < \|f_T(\bar{U})\|_\infty$. Let j be the last iteration such that $\|f_T(\bar{U})\|_\infty \geq \|f_T(\bar{V}^j)\|_\infty$. Let V_a^j be chosen for decrementation. Then $V_i^{j+1} = V_i^j$, $i \neq a$ and $V_a^{j+1} = V_a^j - 1$. Moreover,

$$\frac{T_a}{V_a^j - 1} = \min\left\{\frac{T_i}{V_i^j - 1}\right\}$$

for all i such that $V_i^j \neq M_i$. By choice

$$\|f_T(\bar{U})\|_\infty < \|f_T(\bar{V}^{j+1})\|_\infty .$$

But since V_a^{j+1} is the only component that has changed,

$$\|f_T(\bar{V}^{j+1})\|_\infty = \frac{T_a}{V_a^j - 1} \text{. This implies}$$

$$\frac{T_i}{V_i^j - 1} \geq \frac{T_a}{V_a^j - 1} > \|f_T(U)\|_\infty \geq \frac{T_i}{U_i} .$$

Table 1 Simulation parameters

Parameter	*Possible values*
Number of processors	4, 8, 16, 32, 64
Number of task classes	3, 6, 9
Microstore load time	500 microseconds
Service time quantum	1 millisecond
Saturation limit	≧ 3. number of processors
Tasks processed	2,500
Delay time	100 milliseconds
Task classes	9 possible

But this means $V_i^j - 1 < U_i$ or $V_i^j \leq U_i$ for all i such that $V_i^j \neq M_i$. But applying Lemma 3.2.5 we must have $V_i^j \leq U_i$ for all i; which in turn implies $\Sigma V_i^j \leq \Sigma U_i$. But there exists at least a $(j + 1)^{st}$ iteration so that $K < \Sigma V_i^j \leq \Sigma U_i = K$ which is a contradiction.

Algorithm W

```
1. $W_i := M_i$, for all $i$.
2. ORD($i$) $:= \frac{T_i}{M_i(M_i + 1)}$, for all $i$.
3. If $\Sigma W_i = K$, stop; otherwise, continue.
4. Find $h$ such that ORD($h$) $= \max\{$ORD($i$)$\}$, for all $i$.
5. $W_h := W_h + 1$; ORD($h$) $:= \frac{T_h}{W_h(W_h + 1)}$; Go to 3.
```

Theorem 3.2.7
Algorithm W generates a minisum configuration under the constraint $W_i \geq M_i$ for all i.

The proof of this theorem parallels exactly the proof of Theorem 2.2.1.

4. A simulation study

A simulation study was undertaken to compare minimax allocation, minisum allocation, and an allocation scheme proposed by Burroughs (Davis, Zucker and Campbell, 1972). Burroughs' allocation method queues all tasks in a single queue. Whenever a processor becomes idle, it accepts the next ready to run task from this task queue, loads the task's associated microprogram and begins processing. It was our opinion that this scheme would tend to cause higher microstore loading rates than either minimax or minisum allocation. The

Table 2 Simulation statistics for the systems which used Burroughs' allocation

Task classes	*Processors*	*Loads per task = ML**	*Total loads*
3	4	·41	1036
	8	·47	1189
	16	·46	1185
6	8	·76	1938
	16	·76	1936
	32	·78	1980
9	16	·82	2100
	32	·83	2107
	64	·77	2047

Table 3 Simulation statistics for the systems which used minimax allocation

Task classes	*Processors*	*Reconf. per Task = R*	*Number of reconf.*	*Loads per task = ML*	*Total loads*	*ML* − ML / R*
3	4	·01	11	·01	26	40
	8	·01	16	·02	56	45
	16	·01	21	·07	172	39
6	8	·09	223	·14	352	6·9
	16	·08	194	·18	455	7·2
	32	·04	109	·12	297	16·5
9	16	·13	318	·27	679	4·2
	32	·07	175	·21	523	8·9
	64	·04	89	·22	537	13·8

Table 4 Simulation statistics for the systems which used minisum allocation

Task classes	*Processors*	*Reconf. per task = R*	*Number of reconf.*	*Loads per task = ML*	*Total loads*	*ML* − ML / R*
3	4	·01	15	·02	38	39·0
	8	·01	27	·06	140	41·0
	16	·01	26	·11	275	35·0
6	8	·09	223	·14	350	6·9
	16	·08	210	·21	513	6·9
	32	·04	107	·19	463	14·8
9	16	·11	271	·25	624	5·2
	32	·07	165	·34	853	7·0
	64	·04	100	·28	700	12·2

simulation was designed to test this hypothesis as well as to provide an opportunity to investigate various reconfiguration scheduling techniques.

The main goal of the simulation was to investigate the relative merits of the algorithms as measured by their respective microstore loading rates. Overhead due to reconfiguration was included in the effectiveness measure. The simulation was driven by a set of parameters summarised in **Table 1** (for a more detailed discussion of these parameters and of the simulation programs see Nessett, 1974).

Some explanations are necessary so that the nature of the parameters are well understood. The number of task classes was kept less than the number of processors for each simulation run (see the discussion on 'removing restriction 2' in Section 3). The microstore load time was held constant since the measure of interest—microstore loads per task serviced—was independent of this value. During a task's processing the processor periodically checks to see if a reconfiguration has begun. The time between these checks is the service time quantum. The saturation limit is the maximum number of tasks active in the system at any one time. Each simulation run terminates after 2,500 tasks have been serviced. It was found that the servicing of 2,500 tasks was sufficient to ensure that the statistical measures had settled down to their appropriate values. So that the initialisation of the system would not bias the statistical measures, a delay time of 100 simulated milliseconds was allowed to elapse before any statistical information was gathered. Ad hoc, but, we believe, reasonable assumptions were made about the various probability distributions (e.g. service time distributions for the various task classes) that were used in the simulation. These distributions, which are described in detail in Nessett (1974), were the same for comparable simulation runs across the three allocation schemes.

Before discussing the simulation results, some comments should be made on how reconfigurations were scheduled for the basic allocation algorithms. At first, reconfigurations were scheduled to occur at fixed intervals of time. It was soon discovered, however, that this method of reconfiguration scheduling would not be acceptable because of high processor idle rates. Instead a scheduling technique was used in which reconfiguration occurs whenever a task class queue becomes empty and as a result a processor assigned to that task class becomes idle. This scheme allows the system to respond to its own needs as they arise.

The results of the simulation are tabulated in **Tables 2, 3, and 4.** The measure of effectiveness used to evaluate the three allocation methods is derived from an inequality. If ML is the microstore loads per task for either the minimax or minisum allocation methods, ML^* is the microstore loads per task for the Burroughs allocation method, R is the reconfigurations per task for either the minimax or minisum allocation methods, RT is the time it takes to calculate a new reconfiguration, and $MSLT$ is the time it takes to load a processor's microstore, the basic allocation methods are more efficient than the Burroughs method whenever

$$\frac{RT}{MSLT} < \frac{ML^* - ML}{R}.$$

The figures in the last column of Tables 3 and 4 indicate that the basic allocation methods are quite attractive for systems with a small number of task classes. For systems with more than three task classes, the basic allocation method's attractiveness increases as the number of processors in the system increases. This is probably due to the increase in the number of active tasks with an increase in the number of processors. This increase results in a lower number of reconfigurations per task.

It remains to assess the relative merits of minimax and minisum allocation. A comparison of Tables 3 and 4 shows that minimax allocation is slightly superior to minisum allocation for most of the system configurations. Furthermore, the reconfigurations per task for both allocation methods are equal for all but one of the configurations. This indicates that minisum allocation using the future task entry prediction method mentioned above does not increase the reconfiguration interval as was hoped.

5. Conclusions

A technique has been presented to allocate processors in a dynamically microprogrammable multiprocessor system. This technique attempts to minimise the overhead due to microstore loading in such a system. The technique incorporates a number of algorithms which were shown to optimise certain measures. Finally, the results of a simulation study were presented which indicate that the allocation technique proposed is more efficient than an allocation scheme suggested by Burroughs for realistic values of microstore load time and time to calculate a new system configuration.

References

Arthanari, T. S. and Mukhopadhyay, A. C. (1971). A Note on a Paper by W. Szwarc, *Naval Research Logistics Quarterly*, Vol. 18, pp. 135-138.

Bauer, H. R. (1972). Subproblem of the MxN Sequencing Problem, Report No. STAN-CS-72-324, Computer Science Department, Stanford University.

Bauer, H. and Stone, H. (1970). The Scheduling of N Tasks with M Operations on Two Processors, Report No. STAN-CS-70-165, Computer Science Department, Stanford University.

Coffman, E. G., Jr. and Graham, R. L. (1972). Optimal Scheduling for Two-Processor Systems, *Acta Informatica*, Vol. 1, pp. 200-213.

Davis, R. L., Zucker, S. and Campbell, C. M. (1972). A Building Block Approach to Multiprocessing, *Proc. of AFIPS* (*SJCC*), Vol. 40, pp. 685-703.

Hu, T. C. (1961). Parallel Sequencing and Assembly Line Problems, *Operations Research*, Vol. 9, pp. 841-848.

Nessett, D. M. (1974). Some Design and Allocation Problems for Dynamically Microprogrammable Multiprocessor Systems, Ph.D. Thesis, Washington State University.

Szwarc, W. (1968). On Some Sequencing Problems, *Naval Research Logistics Quarterly*, Vol. 15, pp. 127-155.

Book reviews

Fourier Transformation and Linear Differential Equations, by Zofia Szmydt, 1977; 502 pages. (*D. Reidel Publishing Co.*, $34.00)

This volume derives from the author's lecture courses in Cracow and Warsaw during the years 1966–74, and is a revised and extended version of the original Polish edition of 1972. Its aim is to serve as a textbook for graduate students who attend a course on partial differential equations in a 'modern' setting. By this we mean that the classical concept of a solution is enlarged to include any *distribution* satisfying the given differential equation; if the data occurring in the fundamental limit problems (i.e. initial- or boundary-value problems) considered are sufficiently regular then every distributional solution is also a classical one. The substantial mathematical foundation needed to achieve the extra generality is provided in the first two chapters, which contain an elegant exposition of the theory of distributions and their Fourier transformation. This takes up nearly half the book and could be used independently as the basis for an introductory course on distributions. In the following chapter basic definitions, concepts and methods for differential equations are presented, while the three final chapters are concerned with the application of the method of Fourier transformation to (i) the wave equation, (ii) the equation of heat conduction and Schrödinger's equation, and (iii) the Laplace, Poisson and Helmholtz equations. The aim here is, broadly speaking, to obtain the distributional counterparts of known classical results.

As the author emphasises in her preface, the book is intended as a monograph and not a course on differential equations, and the reader is referred elsewhere for the many important topics omitted. However, for what it achieves in its own particular domain this book can be warmly recommended. Material drawn from many sources (notably the work of L. Schwartz, L. Hörmander and the author's own research papers) is here beautifully integrated and re-presented. No labour has been spared to ensure accuracy, rigour, completeness and intelligibility. There are numerous exercises with hints for solution, a wealth of explanatory footnotes, a list of symbols and an extensive bibliography. The translation is of the highest quality. Except for the unfortunate omission of the publisher's name from many of the references, it is difficult to find a blemish.

G. F. Miller (Teddington)

FORTRAN Programming—A Supplement for Calculus Courses, by William R. Fullar, 1977; 145 pages. (*Springer-Verlag*, $6.80)

This careful book is an introduction to FORTRAN programming intended for use by students who are concurrently taking a beginning calculus course. The aim is to provide a companion to calculus courses so that students can gain added insight into the ideas of the calculus through programming, but most FORTRAN features are fully covered. Calculus topics include sequences and series, numerical integration and elementary differential equations. The author is to be commended on an imaginative approach to the use of computers in mathematics teaching that others might do well to imitate.

Peter Wallis (Bath)