Some Resource Allocation Policies in a Multi Associative Processor *

by

Gary J. Nutt
Department of Computer Science
University of Colorado
Boulder, Colorado 80302

Report #CU-CS-069-75 May, 1975

* This work was supported by the National Science Foundation under grant number GJ-42251.

ABSTRACT

Some Resource Allocation Policies in a Multi Associative Processor

Gary J. Nutt

The Multi Associative Processor is a hypothetical machine composed of eight control units and an arbitrary number of processing elements. Each control unit executes a single-instruction-stream-multiple-data-stream program in conjunction with a subset of the dynamically allocatable processing elements. In this machine, the data bus interconnecting control units and processing elements is partitioned in order to decrease the hardware cost of the interconnection. In previous work, the degree of partitioning was investigated, and it was found that the processing element allocation algorithm was very critical to the performance of the system, [4]. In this paper, a basic allocation algorithm is investigated under various job loads, and then the performance is compared to three distinct algorithms that compact processing elements into smaller subsets of the partitions under varying conditions.

INTRODUCTION

The Multi Associative Processor (MAP) system is an array processor consisting of eight control units and an arbitrary number of dynamically allocatable processing elements, [3]. The architecture has been designed to allow for the simultaneous execution of eight single-instruction-stream-multiple-data-stream programs, where each program uses as many processing elements (PEs) as required [2]; the maximum number of PEs allocated to a given control unit (CU) executing a program depends only on the number of PEs available in the entire system and the requirements of the program, not on a priori limits for each CU. Thus, PEs are dynamically allocatable to each CU from a common pool.

The other primary components of the machine are shown in Figure 1. The input/output subsystem is unspecified at the present time, although it is conceived as being a somewhat conventional subsystem that handles peripheral devices and provides for the transmission of data to and from the main memory system. There are eight memory modules shown in the figure; some preliminary studies into potential memory conflicts have indicated that the system would require at least one memory module per CU, [4]. The memory modules must all be accessible to each CU and to the I/O subsystem; the interconnection of all CUs and all memories is useful to provide for the possibility of shared programs and data, especially if a distributed operating system should be implemented on the hardware as sketched here.

The Distribution Switch shown in Figure 1 is used to route low level commands from the CUs to their respective subsets of PEs. More detail of this switch is provided below, since it contributes significantly to any PE allocation algorithm to be employed in MAP. Each PE consists of an arithmetic/logic unit, a local memory for storing data, and an associative unit. Although the architectural details of MAP are discussed elsewhere, it is noted in passing that the arithmetic/logic unit is similar to a 16-bit minicomputer CPU [1]; the memory size is assumed to be on the order of 1000 words. The associative unit contains an 8-bit "select register" in which individual bits are set/reset under programmer control. This select register setting, along with a key broadcast (implicitly) with each instruction, determines the activity or inactivity of the PE; thus the term "associative" (i.e., content addressable) in the machine's name.

The most critical component of the MAP architecture is the distribution switch for passing control unit commands and data from any control unit to any subset of the processing elements, and also to allow data paths between subsets of PEs. The distribution mechanism consists of two distinct bus systems as indicated in Figure 2; the dashed lines represent possible connections between each CU and each PE, for the purpose of command transmission. In the MAP system, each CU is a microprogrammable control unit that broadcasts 6-bit microinstructions rather than full machine instructions. This allows the instruction bus to be quite narrow, but it requires that it be used frequently. Crossbar conflicts are avoided at the microinstruction broadcast time by fixing the crossbar connection points at PE allocation time; i.e., PE sharing at the microinstruction level is not permitted.

The other portion of the bus system is used for transmitting 32-bit operands or PE memory addresses among CUs and PEs. Reflecting on the architecture as discussed up to this point, it is apparent that the data bus system will be used much less frequently than the instruction bus.

There are two reasons for this: First, not all microinstructions require the data bus, (in fact, it is used primarily for PE address broadcasting and very little for data broadcasting). Secondly, it is expected that for the majority of processing time, operand references will be to the processing element memory rather than the main memory, and for the MAP programs that have been monitored on interpreters, experience has borne out this hypothesis. As indicated in Figure 2, the data bus has been partitioned into j sectors, each sector servicing the needs of k PEs. The conjecture for this design was that it could be successful due to the relatively low load on the data bus. In a previous paper, [4], this conjecture is explored and those results are summarized below.

In order to investigate the viability of partitioning the data bus into j sectors, each sector shared among k PEs, a simulation program was written to model the activity of each control unit at the data bus reference level, i.e., each allocation and deallocation of the data bus for transmission was modeled. To produce a realistic load on the model, it was driven by a full trace of each PE activation, deactivation, allocation and deallocation, while the actual bus interreference pattern was specified by a distribution derived from MAP program executions. The traces and distributions were obtained by interpretively executing each MAP program, one-at-a-time, on a Control Data 6400 and then using the data from the collective runs to drive the model.

The conclusions from this study can be quickly summarized: Even for very heavy loads on the bus system, the number of partitions need not be greater than 8-12 sectors. Secondly, the algorithm used for allocating processing elements to control units is extremely critical to the performance of the system. It is this latter observation which has caused the study that is reported on in the remainder of this paper. Four resource allocation algorithms are examined under a fixed load to investigate some tradeoffs of operating system overhead for more effective utilization of the shared bus system. The hardware is configured to consist of eight sectors in all cases.

The generalization of this problem corresponds to a special crossbar switch. Suppose that a system is composed of n subjects and m objects, and it is required that any subject be able to communicate with any subset of the m objects under the constraint that the communication cannot take place unless all objects in the given subset are "available" simultaneously. If any object is involved in a communication with another subject, then the first subject cannot communicate even with the remaining unencumbered objects.

THE PE ALLOCATION MODEL

The processing element allocation model is based on the same program that was used to investigate the number of sectors in a MAP system, (briefly described above). The primary alteration was to decrease the level of detail to ignore actual bus allocations and deallocations. The reason for this decrease in detail has to do with the higher cost of executing the model at the more detailed level. It appeared that the additional information was not worth the price in terms of execution and programming time. The measure used for bus sector activity in the model is the number of CUs which are allocated active PEs within a given sector, and the number of sectors with active PEs allocated to a given CU. These measures offer an upper bound to, (and is closely correlated with), the actual bus utilization.

Since trace data is not used for determining PE activity, a statistical method must be chosen to generate a set of PE activations, deactivations, allocations, and deallocations. Additionally, the members of the subsets that change from one state to another must be specified. In the previous work, it was observed that MAP programs tend to partition the set of PEs into equivalence classes such that all members in the same class participate in the same action at the same time. In Figure 3, a state diagram is given that indicates all possible states that an equivalence class of PEs can assume. In the simulation model, the state diagram is used to describe PE activity by specifying a distribution of the amount of time a class of PEs will remain in any given state and the transition probability for moving from any one state to another. The resource allocation model allows each CU to be characterized by specifying:

The maximum number of PEs to be allocated to the CU
(maximum $\leq$ 256).

The number of equivalence classes of PEs assigned to the CU
(number $\leq$ 26).

The number and distribution of PEs in each equivalence class.

Transition probabilities and distributions for the state diagram
shown in Figure 3.

The simulation of the system is terminated after simulated time has reached some value provided as an input parameter. A report is produced which summarizes the machine configuration, the algorithm, the CU characteristic as described above, CU-PE and CU-sector allocation status at termination, PE allocation request queue, and a series of histograms to describe sector activity, allocation, number of active CUs per sector and the number of active sectors per CU.

Although the above description of the model is brief, the program itself is relatively complex, and is written in about 1500 lines of FORTRAN and assembly language code.

The model can be criticized on the following points: The character of the programs that run on a given CU is stated with respect to the initial description, i.e., the model does not simulate any form of multi programming of the control units, nor does it allow a job to "finish" and a new job with a new set of characteristics to be loaded onto the control unit. Because of this problem, only non-preemptive resource allocation algorithms are tested on the model. A more complete study would allow preemptive resource allocation.

THE ALLOCATION ALGORITHMS

Four algorithms were implemented and tested for allocating PEs to control units. As noted previously, all of the algorithms are non-preemptive, and three of them use compaction techniques for changing the PEs allocated to a CU from one sector to another. In the MAP system, these compactions require a certain amount of operating system overhead to allow the PE memory contents of one PE to be copied over the data bus system to another PE. Special "stream instructions" exist in the MAP instruction repertoire to rapidly move a block of words. At any rate, the number of PEs moved during any compaction operation is a measure of the overhead that must be offset by any decrease of sector conflict resulting from the compaction.

The basic algorithm for PE allocation to control unit number i, (denoted CU_i), does not use compaction.

1. Test all sectors with the following: If the sector contains PEs already allocated to CU_i and no other CU, then allocate any available PEs within the sector to CU_i. If outstanding requests still exist after testing all sectors, go to Step 2.
2. Test all sectors with the following: If no PEs are allocated within the sector, then allocate to CU_i. If outstanding requests still exist, go to Step 3.
3. Test all sectors with the following: If the sector contains any PE allocated to CU_i and there exist unallocated PEs, then allocate to CU_i. If outstanding requests still exist, go to Step 4.
4. Allocate any available PEs to CU_i, regardless of sector membership.

This basic algorithm attempts to allocate processing elements in the best possible way, given a static set of conditions. Of course conditions are <u>not</u> static in the system, since the eight CUs will be asynchronously allocating and deallocating PEs. The second algorithm takes the dynamic nature into account by attempting to compact after any PE deallocation. Allocation is handled exactly as in the basic algorithm. The basic compaction algorithm consists of:

1. Find a sector that is neither entirely empty nor entirely full (call it sector j), and go to Step 2. If no such sector exists, do not perform a compaction.*
2. If sector j does not contain PEs allocated to at least two distinct CUs, return to Step 1, to choose another sector j. Otherwise, choose the CU that contains the most PEs in the given sector (call it CU_k), and go to Step 3.
3. Search all sectors distinct from sector j such that a new sector (call it sector i), is found that contains PEs allocated to CU_k. If no such sector exists, repeat steps 1 and 2 to find a new CU_k and sector j. If no possible combination of CU, yielding sector, and receiving sector exist, then no compaction can be done. Otherwise, go to Step 4.
4. If the number of PEs allocated to CU_k in sector i exceeds the number of idle PEs in sector j, move PEs from sector i to sector j until there are no more unallocated PEs in sector j, then stop. Otherwise, move all PEs allocated to CU_k from sector

* A refinement to the basic compaction algorithm would attempt to reduce the number of shared sectors if some sector is found to be completely empty.

i to sector j and then go to Step 3.

The remaining two algorithms are simple extensions to the basic compaction algorithm, and differ from the basic algorithm (and from each other) in the conditions under which each will attempt to perform a compaction.

Let M be the number of sectors in the machine and k be an input parameter to the model. Then algorithm number three invokes the compaction algorithm after any deallocation such that the sum of the number of sectors, s_i, allocated to each CU exceeds kM, i.e., $\sum_{i=1}^{8} s_i > kM$. The criterion used here is that compaction should be attempted only when the level of sharing surpasses some absolute threshold.

Algorithm 4 is based on a slightly more sophisticated measure of sector utilization. Let n_i be the number of PEs allocated to CU_i, and m be the number of PEs per sector. Then

$$\left\lceil \frac{n_i}{m} \right\rceil$$

is the minimum number of sectors required under optimal resource allocation. Again let S_i be the number of sectors allocated to CU_i and k be an input parameter, then if

$$\sum_{i=1}^{8} S_i > k\sum_{i=1}^{8} \left\lceil \frac{n_i}{m} \right\rceil$$

algorithm number 4 will attempt compaction after PE deallocation. In the case that k = 1, the algorithm attempts compaction if the current allocation is less than optimal, thus it is likely that the performance of the algorithm would not be any different from the basic compaction algorithm. For k > 1, the amount of deviation from optimal allocation can be used to invoke compaction.

TESTING THE ALGORITHMS

First, a fixed job load is used for comparing performance and cost variations for varying factor k within Algorithm 3 and within Algorithm 4, and then to compare their performance with the basic algorithm and the basic compaction algorithm. Next, a brief description of tests made with different job loads is given.

For testing compaction factors and comparing algorithms, the eight programs whose characteristics are given in Table 1 were used. The maximum number of PEs associated with a program on a CU is initially allocated to the CU, and then individual clusters, (equivalence classes of PEs), are allowed to become active, or be deallocated, as the transition probability for state 1 of the program dictates (see Figure 3). The Table also indicates the number of partitions that should exist within each program; the actual input to the program allows a precise definition of these equivalence classes. The two probabilities, for an activation transition from State 1 or a deactivation transition from State 2, also imply the probability of a deallocation transition from either State 1 or State 2, since each state has two output transitions. Once a partition of PEs enters a state, the "delay time" distributions are used to indicate the amount of time a partition should stay in the given state. The distributions used here were all Erlang distributions, although the model allows negative exponential, hyper-exponential, normal, and uniform distributions, (which were used in other job load descriptions).

Table 2 summarizes some of the results of the tests with the fixed job load; the mean number of CUs and the maximum number of CUs that contained activated PEs in each sector are listed. Algorithm 3 was also tested with k = 1.0 and 1.25, but produced exactly the same result as the basic compaction (hereafter BCA). Similarly, Algorithm 4 with $k = 1.0$ produced the same result as BCA. Since Table 2 ignores the overhead time due to compaction, one can compare pure performance without regard to cost. Considering Algorithm 3, a factor of 1.50 performs as well, on the average, as the compaction on every deallocation (i.e., BCA). If the overhead is less, then Algorithm 3 with $k = 1.50$ (denoted #3/1.50) would be more attractive. Over a simulated time interval of 5000 time units, BCA performed 75 compactions, where the mean number of sector pairs involved was 2 per compaction; the mean number of PEs moved per compaction was 15.9. On the other hand, #3/1.50 performed only 53 compactions, each involving 2.7 sector pairs on the average, and moved an average of 21.5 PEs per compaction. Thus #3/1.50 requires significantly less overhead and still performs as well as BCA. Both are much better than the basic algorithm, BA. #3/1.75 compacted only 20 times over the same time interval; the average number of sector pairs per compaction was 3.8, although the mean number of PEs was only 19.2. Overhead reduction in #3/2.0 was insignificant compared to #3/1.75. It is apparent that for this fixed job load, (see Table 1), Algorithm 3 with $1.50 \leq k \leq 1.75$ is a good choice, with smaller k for pure performance objectives and larger k for less system overhead.

Algorithm 4 represents reasonable performance for $k = 1.25$ (denoted #4/1.25) and $k = 1.50$, (i.e., #4/1.50), although for the given job load, #3/k with $1.50 \leq k \leq 2.00$ all perform as well or better. #4/1.25 performed only 6 compactions during the 5000 units of time and used an average of 4.5 sector pairs per compaction. These compactions were relatively inexpensive, since the mean number of PEs moved per compaction was only 11.5; (in two cases, about 50 PEs were moved on one compaction). #4/1.50

required nearly the same system overhead. The conclusion is that #4/k, for $1.25 \leq k \leq 1.50$ is even more economical than #3/1.75 and it performs almost as well.

The basis for comparing performance above was the number of CUs per sector; another measure of CU blockage due to sector conflict is the number of sectors associated with each CU such that the sector contains at least one active PE allocated to the CU. Table 3 summarizes the simulation results for the fixed job load tests. Because of sector size and the varying number of active PEs per CU, it is difficult to interpret the data except by comparison. If all PEs were active all of the time, the minimum number of sectors required is given in parentheses following the CU number, and the percent of time that the CU used more than the minimum allocation is included in parentheses under each algorithm heading. Again, #3/1.50 performs comparably to BCA and both are much better than BA; again the small differences between #3/1.50 and #3/1.75 are apparent. #4/1.25 and #4/1.50 remain similar.

In the remaining discussion, only BA, BCA, #3/1.50, #3/1.75, and #4/1.25 are used to compare performance on different loads. The following basis of comparison will be used: The state of a PE is defined by Figure 3, and that model corresponds to a Markov chain. Therefore, the steady state probability that a PE is in State 2, (i.e., allocated and active) can be determined from the input parameters to the model; denote this probability as q. An estimate for the expected number of PEs that are active for CU_i is

$$qn_i$$

where n_i is the maximum number of PEs allocated to CU_i. Thus, the minimum expected number of sectors containing active PEs allocated to

CU_i is

$$S_{min}(i) = q + \frac{q(n_{i-1})}{m}$$

where m is the number of PEs per sector. (Note that one sector is required if any PE is active.) Let $S_{exp}(i)$ be the experimental average number of sectors containing PEs allocated to CU_i, and let

$$W_i = \frac{n_i}{\sum_{j=1}^{8} n_j} \quad .$$

Now, a single number representing the performance of an algorithm on a given load can be computed as

$$L_k = \sum_{i=1}^{8} \left(\frac{S_{min}(i)}{S_{exp}(i)} \right) W_i$$

for job load number k.

The simplest meaningful measure of cost, C_k, is the total number of PEs moved during a simulation run on load number k, (which is always zero for BA). This measure does ignore the number of PEs moved per compaction, which provides definite insight into cost.

Figures 4 and 5 illustrate the performance and cost, respectively, of BA, BCA, #3/1.50, #3/1.75, and #4/1.25 on a few different job loads. Although the data has no meaning between discrete points on the abscissa, it is more convenient to use a continuous line for comparing results than a set of bar graphs. The results of the different job loads are reasonably consistent with the fixed job load case discussed above, (job load 0 is that fixed job load). The most apparent anomaly is the cost of running #3/1.50 on load 2; the explanation for this extremely low cost compared to the other compaction algorithms is that an "early" compaction just happened to catch the configuration in such a way that the compaction allocation was very near optimal. Also, the pattern of deallocations did not disturb the fortunate allocation state once it had been reached. (These remarks are substantiated by the data taken on that run).

The conclusion drawn from these tests runs, (and other runs not shown), is that the most cost-effective algorithms are 3/1.75 and 4/1.25, with Algorithm 3 having a slight performance edge but also costing a little more to use.

REFERENCES

[1] Arnold, R. D. and Nutt, G.J., "The Architecture of a Multi Associative Processor", unpublished.

[2] Flynn, M. J., "Some Computer Organizations and Their Effectiveness", IEEE Transactions on Computers, Vol. C-21, No. 9 (Sept., 1972), pp. 948-960.

[3] Nutt, G.J., "An Overview of a Multi Associative Processor Study", Proceedings of the 1974 National ACM Conference, (1974), pp. 101-104.

[4] Nutt, G.J., "Some Considerations of a Certain Critical Component of an Array Processor", submitted for publication.

Program Number	Maximum Number of PEs	Number of Partitions	Probability of activation from State 1	Probability of deactivation from State 1	Delay Time Distribution Mean/Std. Deviation		
					State 1	State 2	State 3
1	100	10	0.95	0.95	25.0/5.0	25.0/5.0	150.0/10.0
2	30	6	0.90	0.90	20.0/2.0	30.0/2.0	100.0/10.0
3	64	10	0.93	0.93	15.0/1.0	15.0/3.0	65.0/5.0
4	15	10	0.96	0.96	10.0/1.0	10.0/1.0	100.0/10.0
5	2	2	0.70	0.70	10.0/3.0	10.0/3.0	30.0/6.0
6	100	10	0.90	0.93	20.0/5.0	30.0/5.0	150.0/10.0
7	32	6	0.85	0.90	29.0/2.0	35.0/2.0	100.0/10.0
8	64	10	0.98	0.98	15.0/1.0	15.0/3.0	75.0/5.0

PROGRAM CHARACTERISTICS

Table 1

Sector	Basic Algorithm	Basic Compaction Algorithm	Algorithm No. 3			Algorithm No. 4			
			k=1.50	k=1.75	k=2.0	k=1.25	k=1.50	k=1.75	k=2.0
1	2.01/4	0.96/1	0.97/2	1.73/4	1.00/4	0.91/1	0.95/1	2.19/5	2.00/5
2	1.61/3	1.38/3	1.29/3	1.07/2	1.99/3	1.27/2	2.05/4	2.34/5	1.86/3
3	2.18/4	2.24/4	2.32/4	2.52/4	2.24/5	3.09/5	1.63/4	2.55/4	2.92/5
4	1.32/2	0.99/1	0.99/1	0.99/1	0.99/1	0.99/1	0.99/1	1.19/2	1.31/2
5	3.42/5	1.33/2	1.31/2	1.24/3	2.24/3	1.37/2	2.24/3	3.69/5	3.53/5
6	3.08/4	2.04/3	1.98/3	2.31/3	2.92/4	2.66/4	3.09/4	3.30/5	3.15/4
7	1.08/2	0.99/2	0.99/2	0.99/2	1.08/2	1.00/2	1.11/2	1.08/2	1.09/2
8	0.82/1	0.98/1	0.98/1	0.91/1	0.90/1	0.99/1	0.89/1	0.88/1	0.79/1
MEAN	1.94/3.13	1.36/2.13	1.35/2.38	1.47/2.50	1.67/2.88	1.54/2.25	1.62/2.50	2.15/4.38	2.08/3.38

Mean/Maximum Number of CUs with Active PEs per sector

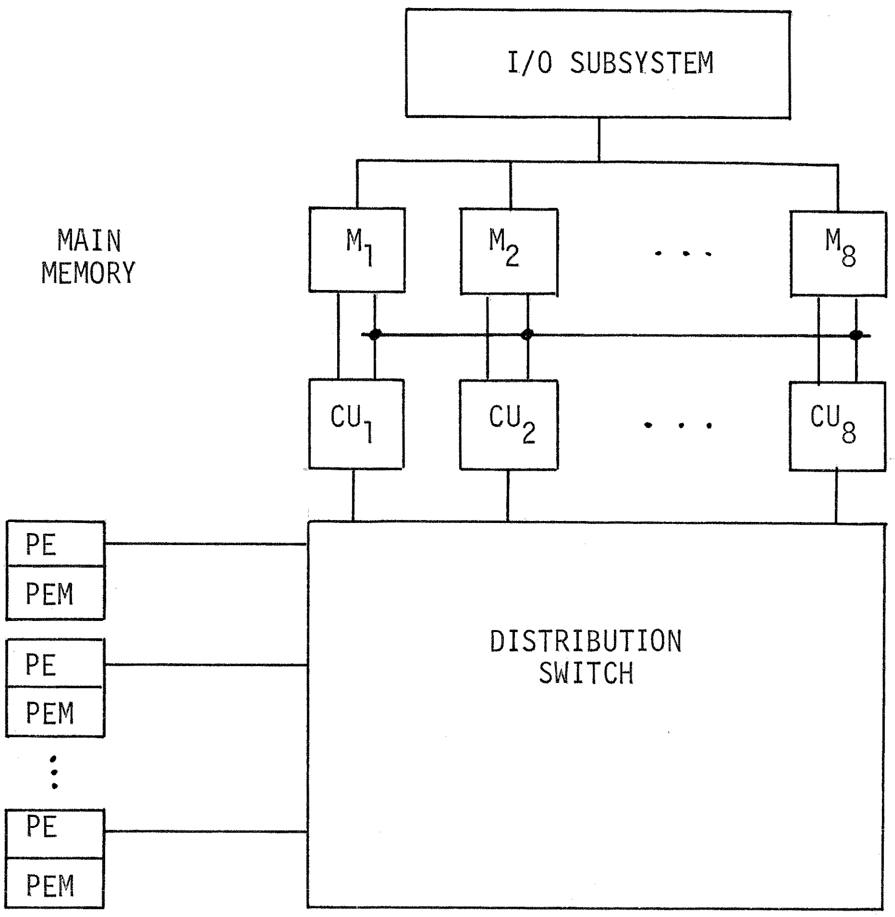
Table 2

CU	Basic Algorithm	Basic Compaction Algorithm	Algorithm 3			Algorithm 4			
			k=1.50	k=1.75	k=2.0	k=1.25	k=1.50	k=1.75	k=2.0
1(2)	3.41/6 (50)	1.97/4 (1)	1.94/2 (0)	2.15/3(22)	2.10/3(31)	2.39/3(48)	2.52/4(32)	2.81/5(49)	3.40/6(46)
2(1)	1.52/3 (55)	0.88/1 (0)	0.88/1 (0)	1.24/2(27)	1.03/2(10)	0.97/1 (0)	0.42/1 (0)	1.95/3(75)	1.36/3(42)
3(2)	2.64/5(23)	1.64/3(16)	1.67/3(16)	1.85/3(11)	2.65/4(55)	1.57/2 (0)	1.74/5 (8)	3.50/5(55)	2.66/5(23)
4(1)	1.15/2(18)	1.03/2 (4)	1.03/2 (4)	1.11/2(12)	0.98/1 (0)	1.08/2 (0)	0.98/1 (0)	1.21/2(23)	1.23/2(25)
5(1)	0.28/1 (0)	0.46/1 (0)	0.44/1 (0)	0.44/2 (2)	0.42/2 (1)	0.53/1 (0)	0.25/1 (0)	0.49/1 (0)	0.29/1 (0)
6(2)	2.67/5(47)	2.19/3(22)	2.19/3(22)	2.12/4(17)	2.85/4(53)	2.74/4(72)	2.80/4(79)	3.32/5(80)	3.30/5(74)
7(1)	2.18/3(75)	0.87/1 (7)	0.88/1 (7)	0.95/2 (5)	1.51/3(51)	1.17/2(32)	2.40/3(85)	2.07/3(76)	2.65/4(90)
8(2)	1.68/2 (0)	1.87/2 (0)	1.81/2 (0)	1.93/2 (0)	1.83/5 (2)	1.84/2 (0)	1.82/2 (0)	1.84/2 (0)	1.73/2 (0)

<u>Mean/Maximum Sectors Containing at Least One Active PE</u>

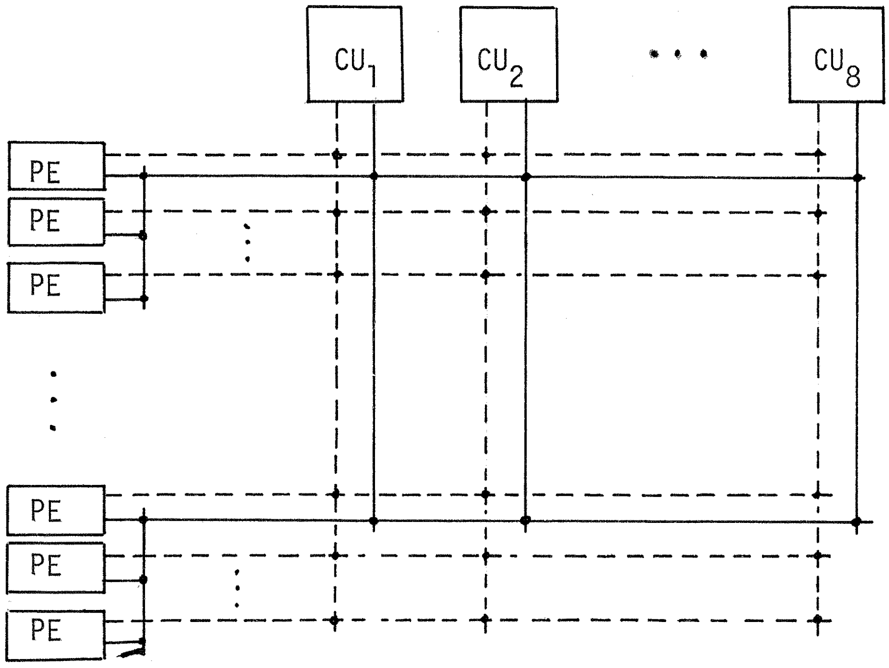
<u>Allocated to a CU</u>

Table 3



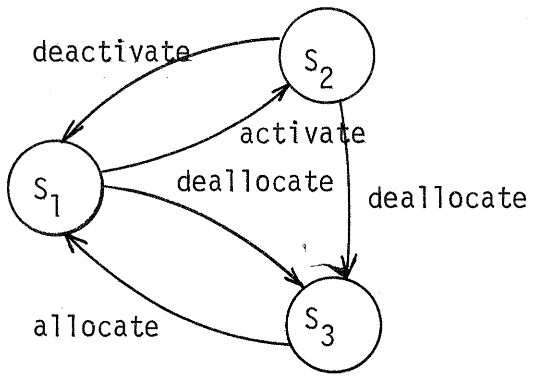
MAP Block Diagram

Figure 1



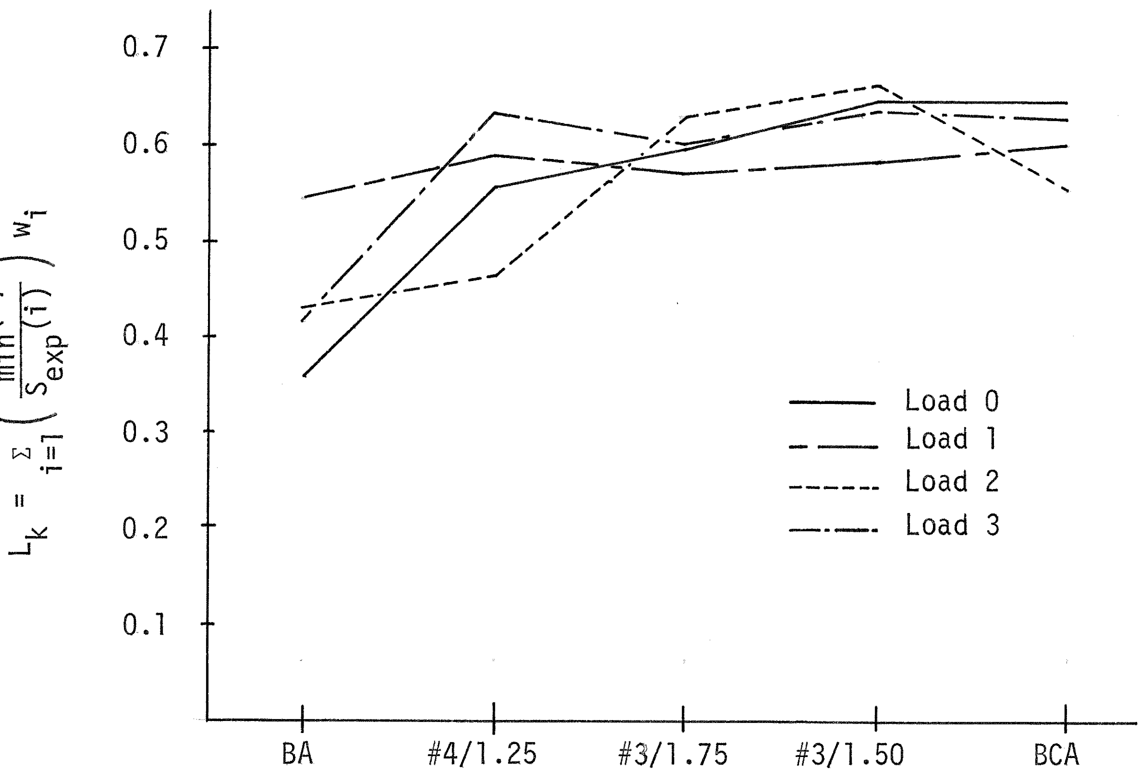
The Distribution Switch

Figure 2



PE State Diagram

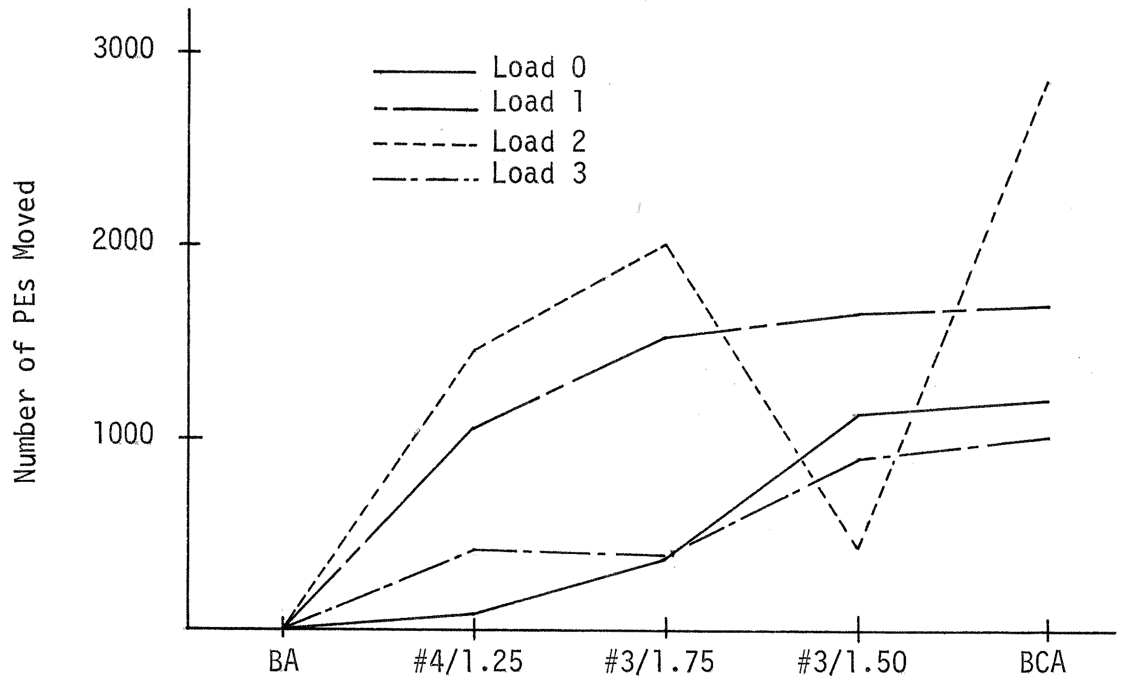
Figure 3



Algorithm Designation

Performance Comparison

Figure 4



Algorithm Designation

Cost Comparison

Figure 5