An Analysis of Multiprocessor Throughput Performance in the Limit†

RUSSELL F. VAUGHAN and MARK S. ANASTAS*

Abstract—This paper describes an analysis of the major sources of overhead in multiprocessor systems with emphasis on performance equations for very large systems. An analytical model was developed for studying the relative contributions of these sources of overhead. Memory contention overhead as modeled is containable within bounds; the limit equations are provided. Software control table lockout, on the other hand, is shown to increase unbounded in large systems in such a way as to limit performance. Methods of effectively reducing overhead from this source are explored. This paper shows that task control efficiency is the only means of achieving efficiency in very large multiprocessor systems. In addition, if such efficiency could be obtained in a centralized control mechanism (by hardware or other means), there would be no other immediate theoretical problems associated with increasing multiprocessor size.

I. INTRODUCTION

Single processor approaches have known limitations to increasing general purpose computer throughput capabilities [10]; moreover, requirements for increased throughput seem more general and insatiable than ever. The advent of inexpensive microprocessors has intensified interest in an effective multiprocessing technology capable of combining many processors to obtain significant throughput. The cost advantages of multi-microprocessors over high speed main frame processors provide a natural motivation for re-evaluating the problems previously encountered in large MIMD multiprocessing systems. Therefore, the central theme of this paper is the limiting performance behavior where many processors are involved.

The theoretical problems associated with deadlock avoidance and synchronizing concurrent processes are well understood [4, 11, 14]. The practical problems, however, which are encountered when implementing large multiprocessing systems, have seemed unavoidable. To address these practical issues, this paper presents a general parameterized model of the major overhead contributions in multiprocessing systems. Descriptions of the individual

†This article, originally entitled "Limiting Multiprocessor Performance Analysis," was reprinted with permission from Proceedings of the 1979 International Conference on Parallel Processing, August 21-24, 1979/Bellaire, MI. © 1979 IEEE.

*Boeing Aerospace Company, P.O. Box 3999, Seattle, Washington 98124

0195-4350/80/1000-0153 $04.14/0

overhead contributions modeled separately are found in the literature, but not the integrated mathematical models presented here. Nor has the emphasis of these other models been on performance expectations in the limit as system size increases. The model described here relates the three major contributions to multiprocessing systems overhead to the desired application program processing requirements in order to assess potential performance capabilities. A diagramatic illustration of the modeled sources of overhead is provided in Fig. 1. These sources are as follows:

1. System Control: the multiprocessor executive control program execution time requirements
2. Control Table Lockout: Common queues are required to provide coordinated control; this implies critical sections in the control program which access these queues.
3. Memory Contention: Common physical memory for multiple processors requires the possibility of multiple processors converging on the same physical memory module, in which case a processor may have to wait until other processors' access requests have been serviced.

To obtain a comprehensive model of multiprocessing overhead, without inappropriate complexity, a hierarchical model was developed. The levels and states in this hierarchy are obvious. Fig. 2 is a diagram of the time expenditure states at the top level of this multiprocessor system model. These states are: P, the normal processor operations associated with instruction sequencing and performing the instructions in its repertoire, and C, the memory delays which may include sequences awaiting memory contention resolution. In order for this model to be valid, both the spatial and temporal distributions of memory access requests must be constant and independent of the changing occupations of the processor. These assumptions are characteristic

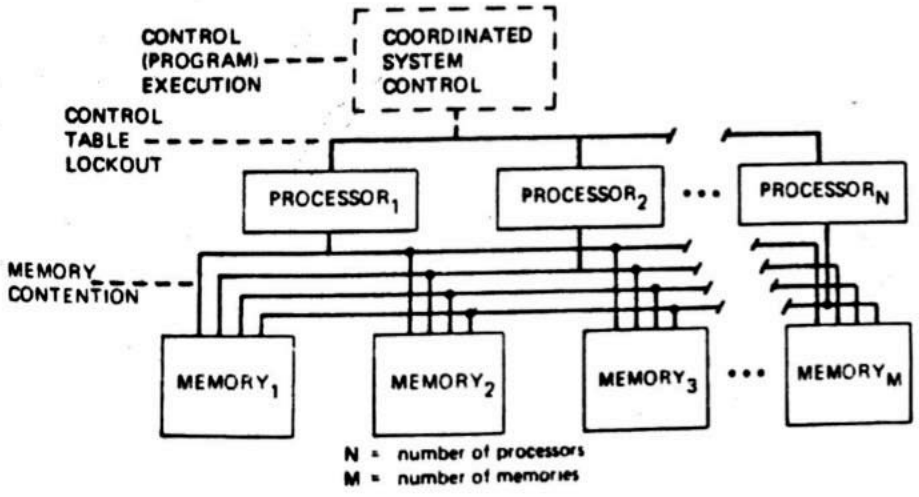

Fig. 1. Modeled sources of overhead

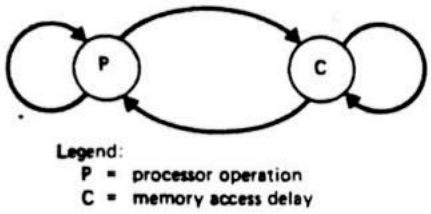

Fig. 2. Time expenditure states

of current multiprocessing. (One of the design trade-offs considered below investigates potential advantages resulting from changing the temporal distribution.)

Time (throughput) overhead is the multiprocessing concern here. Other aspects of multiprocessing, including memory and peripheral sizing, have been modeled in [7]. These other aspects are very important in a system and should be optimized to obtain the best performance for any given configuration. However, they are not the major obstacles to a viable multiprocessing capability.

II. PROCESSOR TIME EXPENDITURE MODEL

The time utility characteristics of the various activities that can be assigned to the processor are modeled here. In a multiprocessor system, the amount of time expended for some of these activities may depend on the number of processors, N. (This definition of N will be assumed throughout the rest of this paper.) The P state of the processor shown in Fig. 2 can be modeled in more detail as shown in the diagram of substates in Fig. 3. The four substates in this diagram are the following:

1. idle state, awaiting an eligible application program task,
2. application program task execution,
3. control program execution, and
4. control table lockout.

In order to get performance predictions independent of the software configuration, it was assumed that the idle state will be null. Here the emphasis is on performance degradation not attributable to insufficient jobs to go around. However, utilization considerations will be discussed below. In addition, lockout is assumed to be experienced only as a part of the control program execution and is therefore called control table lockout. Critical sections in the application program are assumed to be resolved by task eligibility considerations handled by the control program. To resolve such conflicts in the application programs is not the direction of high performance multiprocessing, since excluding the parallel execution of such programs improves throughput. The time line in Fig. 4 shows the phasing among the remaining

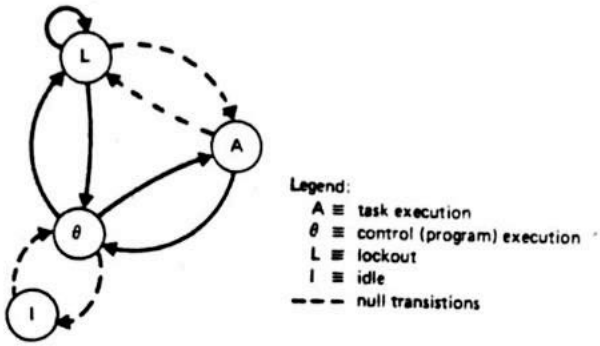

Fig. 3. Second level: time expenditure substates within the processor time expenditure state P

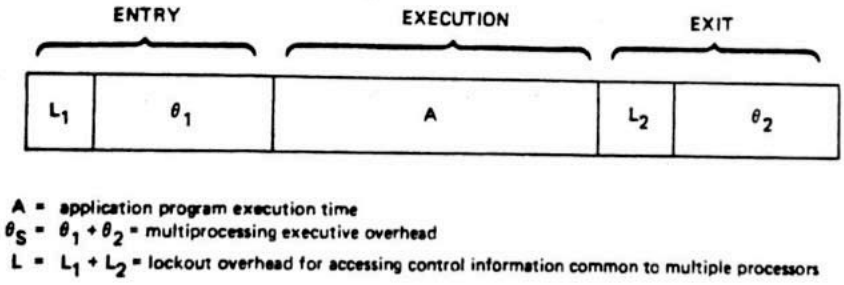

Fig. 4. Task timeline basis for processor overhead

three states. Each of the three remaining processor time expenditures is modeled very simply. An equilibrium situation is assumed among the states, so that the numbers of processors entering and leaving each state are approximately equal. The level of sophistication could obviously be increased in these models, but performance predictions are relatively insensitive to such improvements. The simpler models are easier to describe and understand and fit existing multiprocessor performance data adequately.

2.1 Application Program Task Execution

The model of application task execution involves a constant execution time requirement, A, for all tasks with given queue/dispatch/exit control program request overhead. The model is still valid for programs making multiple requests as long as the ratio of application to control program execution times, $\rho \equiv A/\theta$, is a constant. This ratio is used extensively later on in the analytical derivation of performance and is called the individual processor efficiency. It is affected only by the control program overhead per application task and is defined so as to exclude the effects of lockout induced by multiple processors.

More recent analysis has shown that the model is still valid even when ρ is given by an arbitrary distribution function, as long as the system can be assumed in equilibrium [19].

2.2 Control Program Execution

The execution time of the control program is assumed to be broken into J partitions. These partitions are assumed to be mutually exclusive critical sections with equal execution frequency as well as execution time, $\emptyset_j$.

$$\emptyset \equiv \sum_{j=1}^{J} \emptyset_j = J\emptyset_j$$

For each task, the control program execution time is assumed to be proportional to task execution time, where ρ is the constant of proportionality. This restriction is lifted in [19], however. In addition, this task control overhead is assumed to be independent of the number of processors in the system. The latter assumption implies that queues are implemented with multiple pointers such that lengthy queues do not result in commensurable amounts of searching to process linked task lists. This seems to be a unilateral approach to sophisticated control programs appropriate to multiprocessing.

2.3 Control Table Lockout

Coordination of the activities of many processors to achieve a single computational objective requires the control program to have common task queues for exploiting the parallel aspects of individual application programs. Control table lockout occurs at entry to each of the J control program partitions, each of which is comprised of a mutually exclusive critical section. The total amount of lost time due to this control table lockout will be

$$L = \sum_{j=1}^{J} L_j,$$

where L_j is the amount of lockout time attributed to the jth critical section.

In order to derive an expression from which a value can be computed for the overhead L, we will define N_j as the number of processors waiting and/or executing the jth critical section in the control program. From this definition it can be seen that the amount of lockout time a processor will experience before entering the jth critical section will be $L_j = N_j\emptyset_j = N_j(\emptyset/J)$. N_j can be determined as the probability P_j of an individual processor being in this jth state times the number of possible competing processors, $N - 1$ in this case. (This determination is justified by mean value analysis and the arriver's-distribution theorem in particular [9]). The probability P_j can be determined

as the proportion of time spent in the jth state to the total amount of time spent by each processor.

$$P_j = \frac{\emptyset_j + L_j}{A + \emptyset + L} = \frac{(1 + N_j)}{J(\rho + 1 + N_j)}$$

Thus, since $N_j = P_j \cdot (N - 1)$, a second order equation for N_j is

$$N_j = \frac{(1 + N_j) \cdot (N - 1)}{J(\rho + 1 + N_j)}$$

The formal solution to this equation is

$$N_j = \frac{N - 1}{2J} - \frac{\rho + 1}{2} + \sqrt{\left[\frac{N - 1}{2J} - \frac{\rho + 1}{2}\right]^2 + \frac{N - 1}{J}}$$

For $J = 1$,

$$L = \emptyset \left[\frac{N - \rho}{2} + \sqrt{\left[\frac{N - \rho}{2}\right]^2 + \rho} - 1\right]$$

The expected number of locked out processors is plotted in Fig. 5 for various values of ρ. These curves are in agreement with Madnick [16] in spite of a very different derivation. The significance of increasing the effective number of partitions in the control program will be discussed later.

2.4 Combining Processor Time Expenditures

The objective of the processor activity modeling has been to obtain insight into the relative amount of time spent by each processor in its A, $\emptyset$, and L states. That is, the total number of equivalent processors in the configuration occupied by each activity is of interest. This assessment can be obtained by establishing the ratios of time spent in each activity to the total of a processor's time spent in all activities. By defining X_A, $X_\emptyset$, and X_L as the respective ratios for the A, $\emptyset$, and L activity states,

$$X_A + X_\emptyset + X_L = \frac{A}{A + \emptyset + L} + \frac{\emptyset}{A + \emptyset + L} + \frac{L}{A + \emptyset + L} = 1$$

Furthermore, the equivalent number of processors involved in each activity in an equilibrium situation N_A, $N_\emptyset$, and N_L can be determined as

$$N_A = X_A \cdot N,\ N_\emptyset = X_\emptyset \cdot N,\ N_L = X_L \cdot N$$

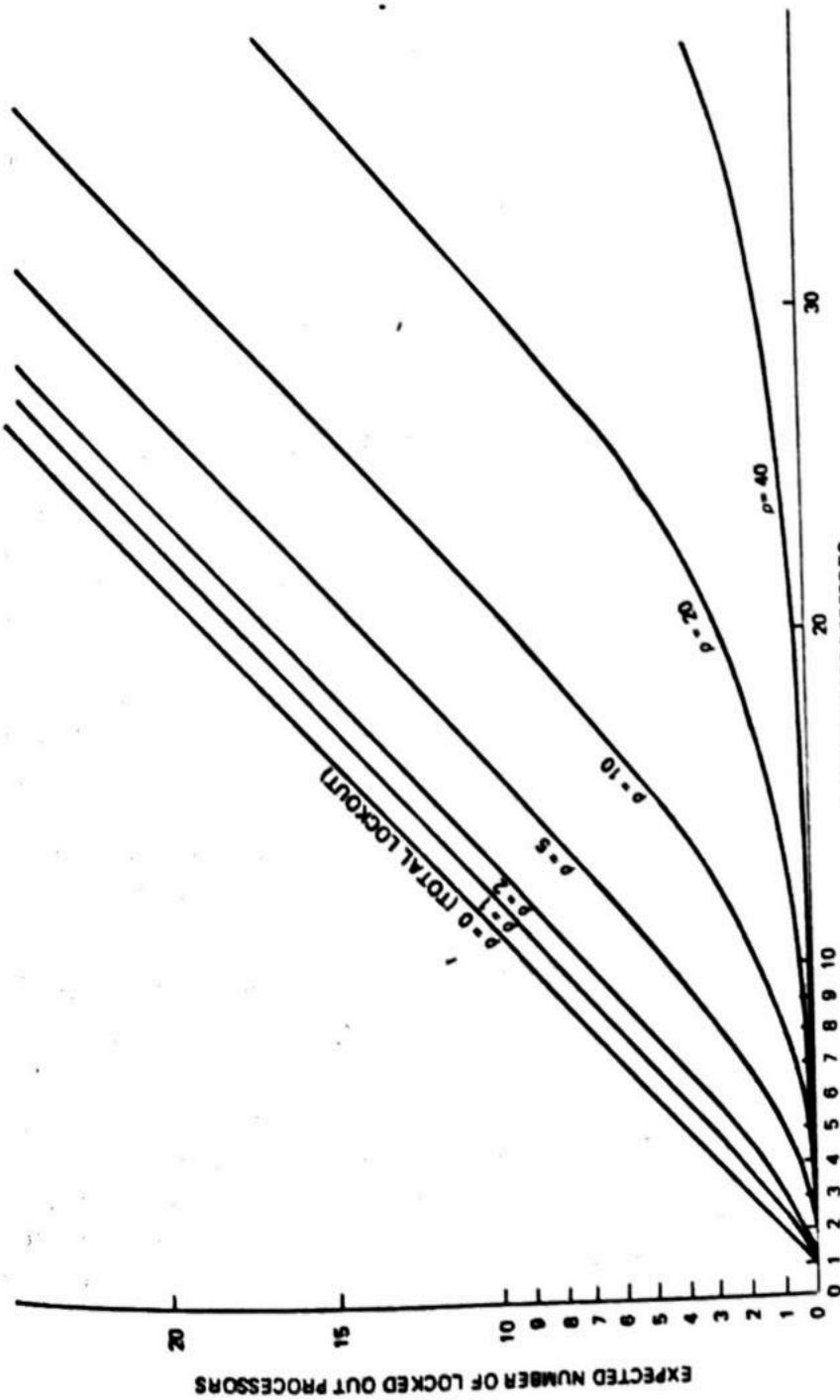

Fig. 5. Impact of processor efficiency on lockout overhead

In order to establish these relative contributions, the results obtained previously can be substituted. The unit of processor time for $J = 1$ is defined as

$$U \equiv \rho + 1 + L/\theta = \frac{N+\rho}{2} + \sqrt{\left[\frac{N-\rho}{2}\right]^2 + \rho}$$

and

$$N_A = \rho \cdot N \cdot U^{-1}$$
$$N_\theta = N \cdot U^{-1}$$
$$N_L = (U - \rho - 1) \cdot N \cdot U^{-1}$$

III. MEMORY CONTENTION DELAY MODEL

Various memory/processor interconnection schemes can be employed for access arbitration, including multiport controllers and crossbar switches as described by Enslow [12] which realize the logical interconnecting paths shown in Fig. 1. Specific configuration dependencies such as processor clock phasing, memory address interleaving, processor to memory speed ratios, and processor memory request duty cycle are discussed in [18]. The mathematical modeling of the performance to be expected of configurations incorporating such dependencies is addressed here.

In a general multiprocessor configuration with M memories and N processors, the percentage of time that the processors spend waiting for a memory to service their requests is important [3, 13].

3.1 General Model of Synchronous Interleaved Memory

To simplify the model, the likelihood of a processor accessing any of the memories on a given request is assumed to be equal. Address interleaving makes this assumption realistic. In addition, each processor presumably makes a synchronous memory access each cycle; this is a worst case situation tending to make the resulting performance predictions pessimistic.

With the probability, $P_S(i)$, of exactly i processors converging on single memories anywhere in the system on a given access,

$$P_S(i) = \sum_{j=1}^{\lfloor N/i \rfloor} p_s(i, j),$$

where $\lfloor x \rfloor$ is the largest integer less than or equal to x, and $p_S(i, j)$ is the probability that there are j instances of exactly i processors converging on single memories in the system. Consider the conditional probabilities $p_p(i, j)$,

and $p_m(i, j)$ which are respectively the probabilities of a processor and a memory being involved in an i-way convergence of processors on memories given that there are j instances of such convergence in the system. Under the assumptions made about random accessing and equivalence between processors,

$p_p(i, j) = i \cdot (j/N)$, since $i \times j$ of the N processors are involved, and

$p_m(i, j) = (j/M)$, since j of the M memories are involved.

Now the unconditional probabilities of processors and memories being involved in i-way convergence situations can be determined as

$$P_p(i) = \sum_{j=1}^{\lfloor N/i \rfloor} p_p(i, j) \cdot p_s(i, j) = \frac{i}{N} \sum_{j=1}^{\lfloor N/i \rfloor} p_s(i, j) \cdot j$$

$$P_m(i) = \sum_{j=1}^{\lfloor N/i \rfloor} p_m(i, j) \cdot p_s(i, j) = \frac{i}{M} \sum_{j=1}^{\lfloor N/i \rfloor} p_s(i, j) \cdot j$$

Therefore,

$$P_p(i) = i \frac{M}{N} P_m(i)$$

3.2 Modeling Memory Response Time

So far only the probabilities of processor/memory convergence have been covered; the real interest lies in contention situations where processor time is lost. Therefore, assume that there is some number, k (not necessarily unity, but for convenience a positive integer), of processors whose requests can be accommodated by each memory module without any of the contending processors experiencing delays. k is defined as the ratio of processor request time over memory response time. A new conditional probability, $P_R(i)$ can be defined which is the probability that a processor involved in an i-way convergence situation will actually experience contention; $P_R(i) = [(i - k)/i]$, for $i > k$; $P_R(i) = 0$, otherwise. The probability, then, that a processor will experience memory contention due to i-way convergency situations is

$$P_C(i) = P_R(i) \cdot P_p(i)$$

$$P_C(i) = (i - k) \cdot \frac{M}{N} P_m(i), \text{ for } i > k;$$

$$P_C(i) = 0, \text{ otherwise.}$$

The total probability of a processor experiencing memory contention P_C can be computed as

$$P_C = \sum_{i=k+1}^{N} P_C(i),$$

since contention can only occur when $i > k$. Therefore,

$$P_C = \frac{M}{N} \sum_{i=k+1}^{N} P_m(i) \cdot (i - k)$$

3.3 Approximating the Distribution Function

Obtaining a distribution function $P_m(i)$ is now required. Many such models of processor queueing on individual memories have been advanced [3, 8]. Little accuracy advantage accrues from selecting the more sophisticated models involving Markov chains. This is particularly applicable for the configurations discussed in this article where memory contention is small, since configurations for which $M \geq N$ and $k \geq 1$ are of primary interest. Bhandarkar [6] has shown errors of less than 5 percent in all cases for the model assumed here.

The model selected is the binomial approximation of Strecker [17], which was found to "work well in all cases" by Baskett and Smith [5], and with more accuracy for $M \geq N$ by Bhandarkar [6]. This model is precisely valid for the initial allocation of processors to memories under the assumptions made previously.

According to this model, the probability that exactly i processors converge on a given memory module in a given cycle is

$$P_m(i) = \binom{N}{i} \left(\frac{1}{M}\right)^i \left(1 - \frac{1}{M}\right)^{N-i}, \text{ where } \binom{N}{i} \equiv \frac{N!}{i!(N-i)!}$$

Therefore, according to this model,

$$P_c = \frac{M}{N} \sum_{i=k+1}^{N} \frac{N!(i-k)}{i!(N-i)!} \left(\frac{1}{M}\right)^i \left(1 - \frac{1}{M}\right)^{N-i}$$

The form of P_c as a function of the number of memory modules is shown for $N = 20$ processors in Fig. 6. The impact of varying the relative speed k of memory access and processor request logic is illustrated in the figure, applying respectively for $k = 1$, 2, and 3.

All of the convergence and contention probabilities are functions of M, N, and k, specifically $P_c = P_c(M, N, k)$. The probability distribution functions P_p and P_m are functions of M and N as well as i, e.g., $P_m(i) = P_m(i, M, N)$.

3.4 Limiting Behavior of "Square" Systems

Note that memory contention decreases very rapidly with M until the numbers of memories and processors are approximately equal ($M = N$), and very slowly thereafter. Systems for which $M = N$ are "square" multiprocessors, and $P_c\,(M = N, k) = P_c(M, M, k) = P_c(N, N, k)$. To understand the significance of configuring multiprocessors with approximately equal numbers of memory modules and processors, consider the limiting values of $P_c(M, N, k)$ as M and N become large. The limits of the summation can be changed to obtain

$$P_c = \frac{M}{N} \sum_{i=0}^{N} (i - k) P_m(i) - \frac{M}{N} \sum_{i=0}^{k} (i - k) P_m(i).$$

Then, since

$$\sum_{i=0}^{N} P_m(i, M, N) = 1,$$

$$P_c = \frac{M}{N} \sum_{i=0}^{N} i\, P_m(i) - k \frac{M}{N} - \frac{M}{N} \sum_{i=0}^{k} (i - k)\, P_m(i).$$

To obtain a limit for P_c, $M = N$ is substituted into $P_m(i, M, N)$ and

$\frac{1}{e} = \underset{N-\infty}{\text{Limit}} \left(1 - \frac{1}{N}\right)^N$. Then, for "square" systems:

$$\underset{N-\infty}{\text{Limit}}\, P_c\,(M = N, k) = 1 - k + \frac{1}{e} \sum_{i=0}^{k} \frac{(k - 1)}{i!}$$

The limiting values for $k = 1$, 2, and 3 are shown in Table I.

3.5 Incorporating Access Duty Cycle

In real systems there is typically not exactly one memory access per processor per request cycle, and the processors are not synchronized relative to whether they actually access memory on a given cycle. There are two typical processor architecture characteristics which affect duty cycle:

1. Processor operations do not typically require an access on every cycle of the instruction. For example, statistically, somewhat less than half of the TI 9900 microprocessor machine cycles require a memory access.
2. Some processors implement a cache memory scheme for look-ahead memory accessing to reduce the average wait time in the processor. This

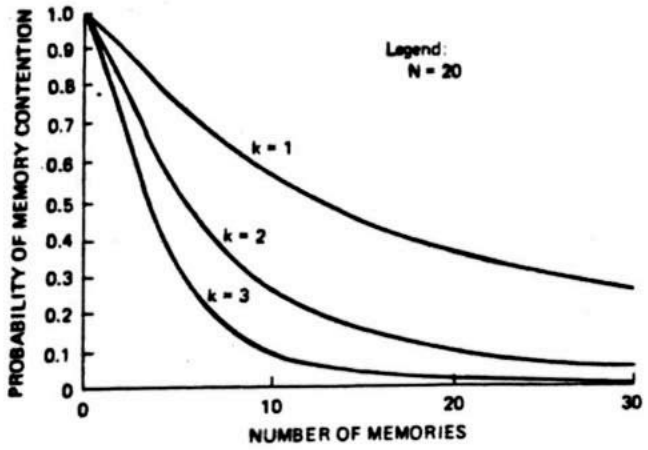

Fig. 6. Impact of ratio of processor request to memory response times

TABLE I
Limiting Memory Contention Probabilities

Relative Speed (Memory to Processor, k)	Asymmetry Ratio (Numbers of Processors to Memories, η)	Limiting Contention Probability (Limit $P_c(M, N))$ $M, N \to \infty$
1	1	0.368
2	1	0.104
3	1	0.023
1	1/2	0.213
1	1/3	0.150

reduces the number of cycles for which the processor makes memory accesses but substantially increases the total number of accesses outstanding when they are made.

These combined phenomena establish an effective, though statistically varying, memory access duty cycle. These characteristics of real systems cannot be modeled by varying the memory to processor speed ratio k. However, at least where large numbers of processors are assumed, an approximately constant access duty cycle d can be expected. This access duty cycle will alter the apparent number of processors actually making memory accesses at any particular cycle to an equilibrium value for large systems of $N = d \cdot N'$. Real "square" systems would then be characterized by the model as "rectangular" systems of dimensions $N = \eta \cdot M$, where η is the apparent asymmetry ratio.

3.6 Limiting Behavior of "Rectangular" Systems

Of interest are memory contention effects when system size is increased in congruent rectangular form. As was the case for "square" systems, for large "rectangular" systems the contention probabilities level off to approximately constant values. Chang, Kuck, and Lawrie [10] derived an expression for the limit from the memory's viewpoint (the probability of a memory rather than a processor being involved in a contention situation). The results do not incorporate the speed ratio k.

Limiting processor contention in large "rectangular" systems can be derived by using the same approach as described previously for "square" systems:

$$\underset{M-\infty}{\text{Limit}}\, P_c\,(N = \eta \cdot M,\, k) = 1 - \frac{k}{\eta} + \frac{1}{e} \sum_{i=0}^{k} \frac{(k-i)\,\eta^{i-1}}{i!}$$

Accuracy considerations relying on Bhandarkar's [6] data suggest $\eta \leq 1$ as the primary domain of usefulness for this equation. The limits for $k = 1$ and for asymmetry values $\eta = 1$, $1/2$, and $1/3$ are shown in Table I. The asymptotic approach to these limits is shown in Fig. 7.

3.7 Combining Processor and Memory Contention Overhead

In the previous accounting of processor time expenditures, only three categories corresponded to the three processor states of application program, control program, and control table lockout. However, not all of the time spent in these three states is correctly attributed to these causes, since memory contention takes a proportional amount of time from each. By this assumption, $C = (A + \emptyset + L) \cdot P_c$. Thus, if the respective primed quantities represent the time in each state exclusive of memory contention, $A + \emptyset + L = A' + \emptyset' + L' + C$ and therefore, $A' + \emptyset' + L' = (A + \emptyset + L)(1 - P_c)$. The respective number of equivalent processors in a multiprocessor configuration expended in the various states are the following:

$$N_A' = N_A\,(1 - P_c)$$
$$N_\emptyset' = N_\emptyset\,(1 - P_c)$$
$$N_L' = N_L\,(1 - P_c)$$
$$N_C = (N_A + N_\emptyset + N_L) \cdot P_C = N \cdot P_C$$

The form of N_C is independent of N_A, $N_\emptyset$, and N_L. The value of N_C increases linearly with increasing system size for congruent rectangular increases, with the slope depending upon the relative speed of the memories and processors and the asymmetry ratio. This phenomenon is shown in Fig.

8. The dashed lines represent the extrapolations from data presented by Bhandarkar [6] which resulted from a more accurate Markov chain model for $k = 1$.

The form of the other three expected numbers of processors N_A', N_θ', and N_L' can be obtained by substitution from previously obtained solutions for N_A, N_θ, N_L, and P_C. It should be clear that N_A' provides a desirable measure of throughput in multiprocessors by providing the effective number of processors being applied to the application programs (the real objective of the system).

To understand the importance of individual processor efficiency on multiprocessor throughput performance, it is interesting to look at the form of $N_A'(\rho)$.

$$N_A' = \frac{\rho \cdot N \cdot (1 - P_C)}{\frac{N + \rho}{2} + \sqrt{\left(\frac{N - \rho}{2}\right)^2 + \rho}}$$

For large N there is an asymptotic approach to a limiting throughput τ, and this limit is

$$\tau = \underset{N \to \infty}{\text{Limit}}\, N_A' = \rho \cdot (1 - P_C)$$

The trailing factor approaches a limit as well, since in general P_c is a function of N. Thus, the control program efficiency not only determines the utili-

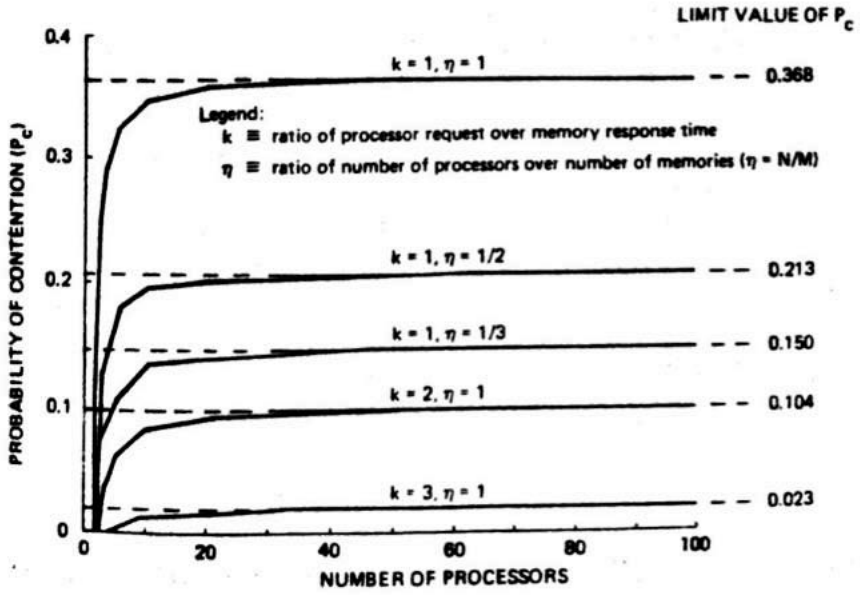

Fig. 7. Asymptotic approach to limiting memory contention probability values

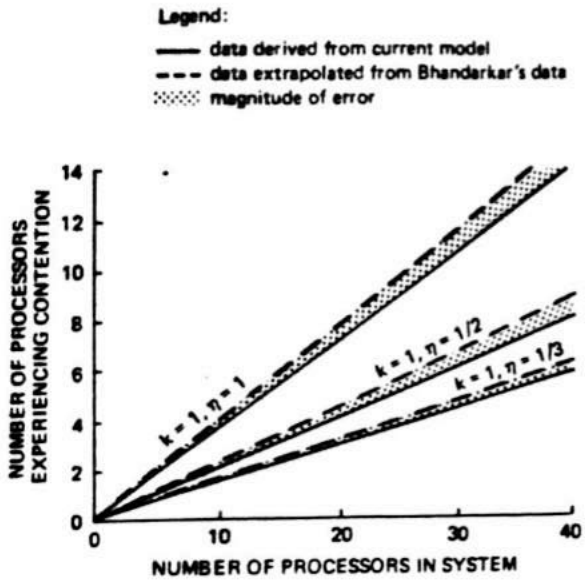

Fig. 8. Number of processors experiencing memory contention in "rectangular" systems

zation per processor, but also determines maximum achievable throughput of the entire machine. In Fig. 9 (which represents state of the art capabilities in large scale multiprocessor systems) there is a maximum achievable return (even with $P_c = 0$) of two equivalent processors applied to application programs. By adding any number of processors beyond 4, the most that will be gained is 0.35 equivalent processors applied to application programs.

The multiprocessor application discussed in [2] is interesting in this regard, however, since for text editing application programs in a multiprogramming timeshare environment, a ρ value as high as 11.5 was obtainable by using an off-the-shelf executive program. The measured performance curves plotted in that paper also fit the predictions of the current model.

The previous limit equation also indicates the impact of memory contention on maximum performance. Memory access efficiency ρ_M, the probability of not experiencing contention on an access, can be defined as follows: $\rho_M = 1 - P_C$. Then, maximum throughput τ for the whole system is equal to the product of the efficiencies of an individual processor ρ_p computed with no contention or lockout, and the ρ_M of the memory accesses:

$$\tau = \rho_P \cdot \rho_M \cdot$$

IV. DESIGN TRADE-OFFS IN MULTIPROCESSORS

In the example shown in Fig. 9, memory contention is not responsible for the reduced efficiency of processors as a function of their increased number.

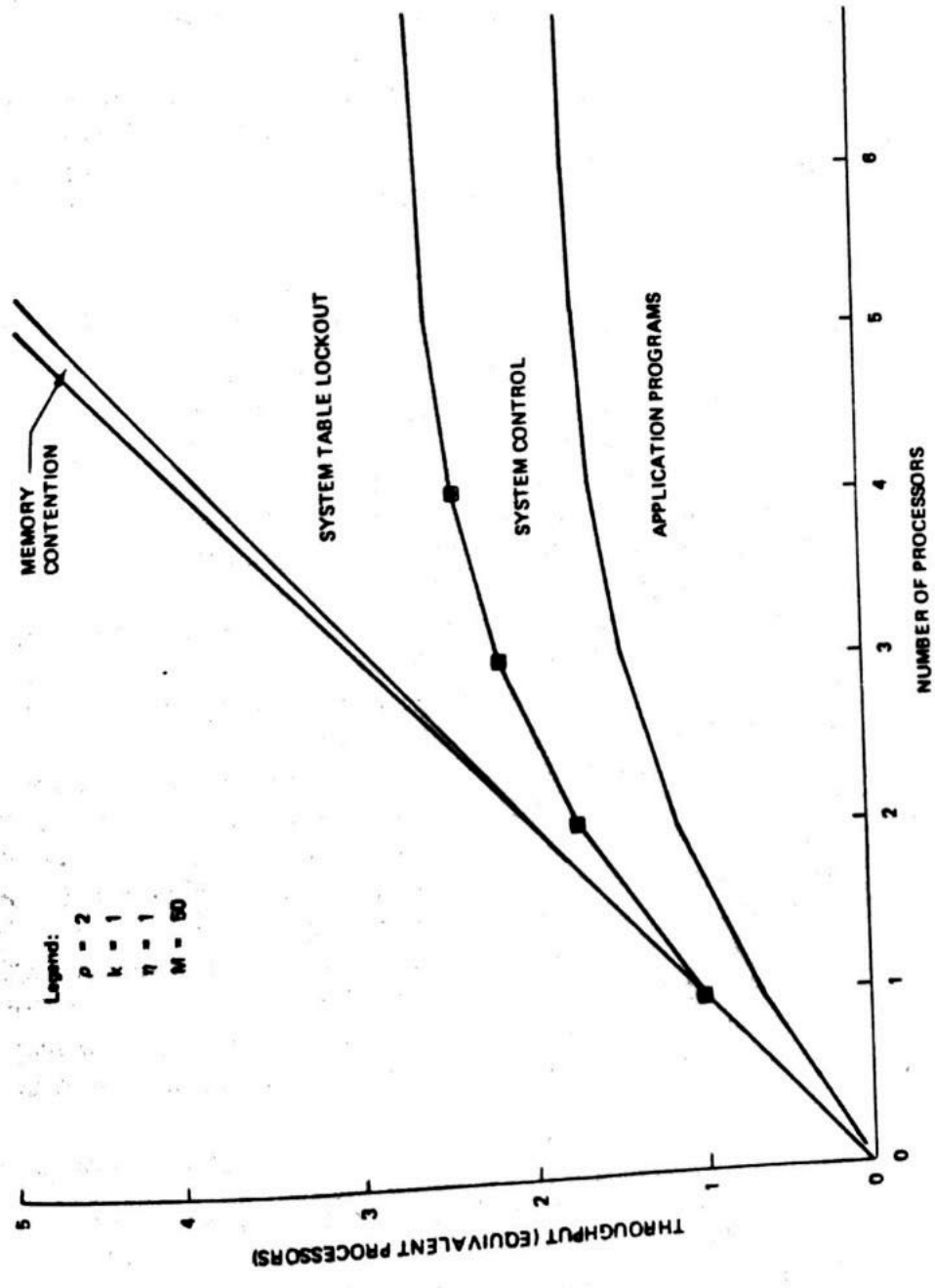

Fig. 9. Overhead contributions

This does not mean that memory contention cannot be a very significant overhead factor, but it is a problem for which solutions exist within current memory and interconnection network technology. In the example of Fig. 9, memory contention is reduced to insignificance by the large number of memory modules ($M \gg N$). Another method that solves the memory contention problem, which is particularly appropriate in microprocessor systems, is increasing the relative speed of the memories. These solutions are appropriate respectively to large mainframe configurations requiring a large memory base to perform their normal operations and to microprocessor-based systems for which obtaining relatively fast memories is not strictly required.

4.1 Reducing Memory Contention

There are, of course, many configurations for which memory contention appears to be very significant. In the solid lines in Fig. 10, the situation previously presented in Fig. 9 has been modified to include only five rather than fifty memory modules. This example actually shows a negative improvement in application program throughput for more than four processors. This negative return can be attributable to the increasing number of processors locked out. These processors are assumed to access semaphores in main memory and thereby contribute heavily to memory contention and are not productive even when successful. This phenomenon can be eliminated by assuming that the semaphores are stored in a special purpose memory dedicated to semaphore control. In this case the ratio of the numbers of processors in the three processor states independent of contention are the same. The effective number of processors competing for memory is reduced, however, to $N_A + N_0$. Estimating P_C for five memories and $N_A + N_0$ processors results in the revised overhead plots shown as dotted lines. The marginal gain in performance for few processors is apparent. Memory contention has been effectively reduced, but the advantage has largely been taken up by increased lockout and system control overhead. This example illustrates the very important point that memory contention can be reduced to insignificance without a commensurable return in throughput. (See also Flores [13].)

4.2 Reducing Processor Lockout

From the preceding discussion, lockout is clearly the primary contributor to multiprocessor inefficiency for large numbers of processors. The starting point of reducing lockout is to consider the assumptions that went into the model of control table lockout. The primary assumption was that the control tables are locked out throughout the execution of the control program. Thus, the approaches in attempting to resolve the control table lockout problems are to

1. design a control program employing a more limited use of lockout,

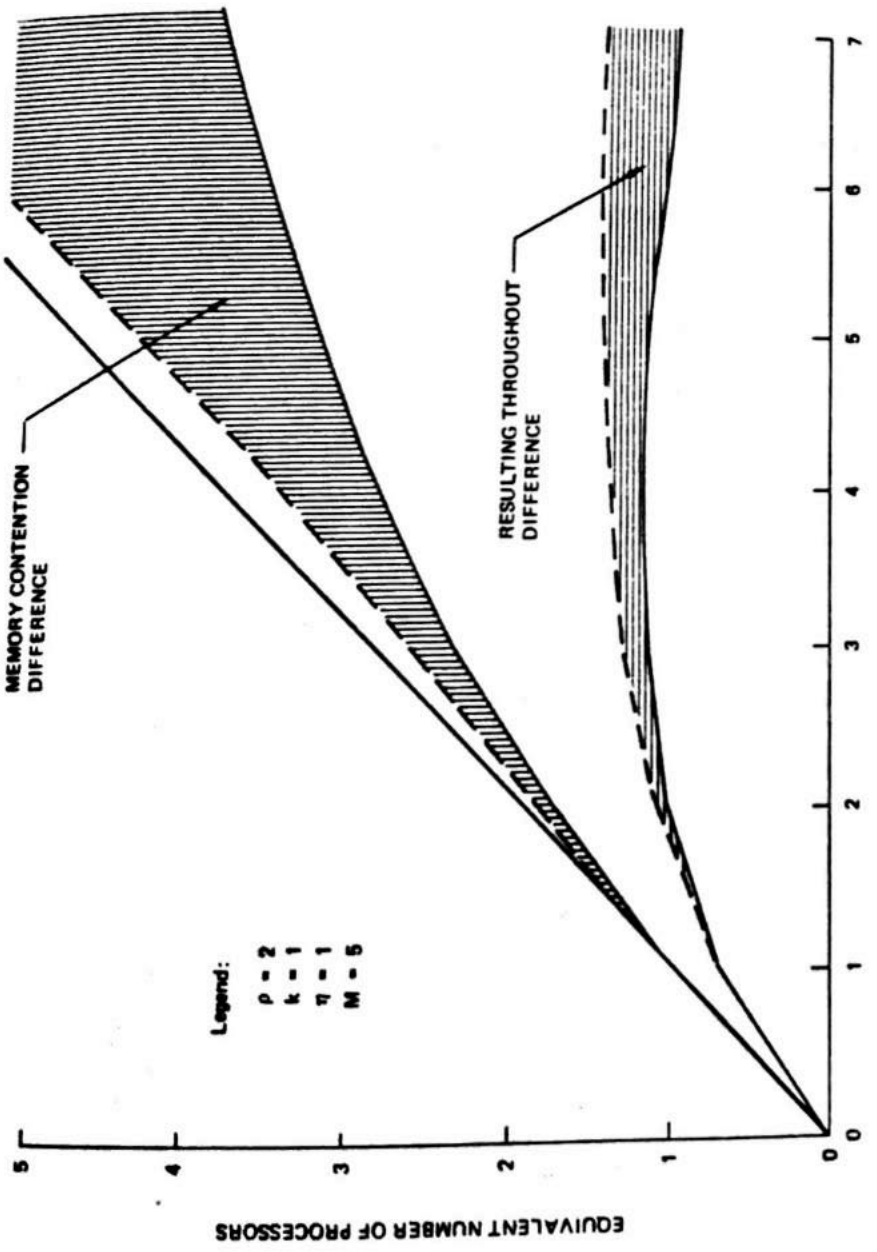

Fig. 10. Ineffectiveness of reducing memory contention

2. reduce the execution time of the critical sections in the control program, and
3. partition the control program into many separate critical sections rather than a single common critical section.

The relative effectiveness of reducing lockout overhead ultimately depends upon the design of the control program itself. There are upper limits for each of the methods. The amount of processing power released to application programs as the result of improvements in these areas will be discussed below. For few processors (small N) the advantage of reducing the length of critical sections or increasing the number of partitions is negligible, whereas an improvement in control program overhead is an immediate advantage even for few processors. For large N the improvement in performance has the same form for reducing extent of critical sections and improving efficiency.

These three solutions have direct analogs in the reduction of memory contention. They are, respectively, reducing accesses to common memory, increasing the relative speed of memory response logic, and increasing the number of independent memory modules which can be accessed. Solutions incorporating the three approaches to lockout are illustrated in the following discussion, the improvements relative to the system whose performance characteristics were shown in Fig. 9. Line A in Fig. 11 represents this baseline system's throughput performance.

4.3 Limiting Control Program Lockout

The entire control program need not be locked out so that only one processor be executing it at any one time. In Fig. 11, line B, the expected perfor-

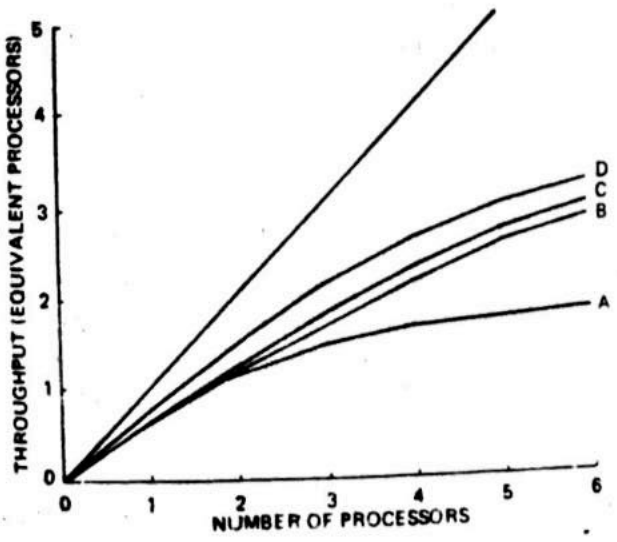

Fig. 11. Comparison of approaches to reducing control table lockout

mance is shown for a system whose control program must be locked out only half of the time. The data for the figure were obtained using a different value of ρ for lockout than for determining the proportion of useful work performed. $\rho_{LOCKOUT}$ can be computed as the total amount of time the processor spends in nonlocked-out processing states divided by the amount of processing time spent in states for which lockout is required. In the current model this can be expressed as

$$\rho_{LOCKOUT} = \frac{A + \emptyset(1 - Z)}{Z\emptyset} = \frac{\rho + 1}{Z} - 1,$$

where Z is the proportion of the control program requiring lockout. In Fig. 11, line B, $\rho = 2$, $Z = 0.5$, and therefore $\rho_{LOCKOUT} = 5$.

4.4 Control Program Efficiency

An obviously effective method of improving multiprocessor throughput would involve directly decreasing the execution time of the critical section portions of the control program. Figure 11, line D illustrates the performance to be expected if the efficiency of the control program were improved by a factor of 2. In this case $\rho = 4$ instead of $\rho = 2$.

4.5 Partitioning the Control Program

The lockout which is necessary in control programs does not necessarily lock out all of the critical sections in the program. Earlier, an equation was developed for lockout that assumed there were J partitions of the control program with independent critical sections. This equation was used to obtain the performance indicated in Fig. 11, line C, for $J = 2$. In [20] it was also suggested that a small number (2, 3, or 4) of partitions significantly improves efficiency. However, it should be obvious that the limiting number of partitions that could be incorporated is not large.

4.6 Increasing Individual Processor Efficiency

The level at which application programs interface with the control program has the same impact on efficiency as the overhead involved in the control program. If the execution time of the typical application program task is increased to the point where the number of executable instructions is doubled, the same efficiency advantage will accrue as if the overhead of the control program were reduced to one half its original value. One must be careful in this regard, however, since the utilization of processors can be significantly reduced. Utilization was ignored in this paper by assuming that there are no processors in the idle state. (See Fig. 3.) The job control languages of batch processing systems largely determine the task level. This is a critical issue

particularly in mainframe multiprocessing but is beyond the scope of this paper.

Kuck [15] investigated the potential for breaking up general problems into parallel segments to obtain commensurable speedup. The inherent parallelism was roughly proportional to the size of the application program if the program units which were dispatched were taken to a low enough level. This finding is in contrast to what was formerly thought to have been an order of log relationship [15]. Thus, the programs themselves have a potential for solution by parallel arrays of slow processors to obtain very high throughput. However, this level would reduce the effective value of A by orders of magnitude, which would in turn reduce ρ (and with it feasibility) by orders of magnitude. Thus, methods which artificially increase ρ do not attack the multiprocessor problem.

V. THE FUTURE OF MULTIPROCESSING

The high leverage design considerations in multiprocessing at this time are control table lockout and the control program overhead. Hardware support for the multiprocessor executive is the obvious place to look for help, since the improvement required to realize large arrays of processors is orders of magnitude rather than simple multiples.

To determine whether there are other theoretical problems, consider Fig. 12. Line A illustrates the system described originally in Fig. 9 but assumes an individual processor efficiency of $\rho = 100$. In this configuration memory, contention becomes appreciable after about 10 or 20 processors, and the maximum achievable throughput is about 40 processors. However, as shown in Fig. 12, line B, the asymptotic limit can be more than doubled by increasing the relative speed of the memory response logic. In this case $k = 3$ rather than $k = 1$.

With such high throughput systems, however, there would be requirements for commensurably larger numbers of memory modules. Fig. 13, line A, illustrates the situation for $\rho = 100$ with "square" multiprocessors ($M = N$) and $k = 1$. The improvement in contention with increasing memory response time can be seen in lines B and C respectively for $k = 2$ and $k = 3$.

VI. CONCLUSIONS

At least analytically there are no size limitations to conventional multiprocessing approaches which are beyond the current state of the art, except control program efficiency. Hardware seems to be the only effective way of significantly improving this parameter. Exploring methods of increasing hardware support for the control programs is the most likely way to extend the limits for multiprocessor throughput performance. The authors are currently exploring such an approach [1, 18].

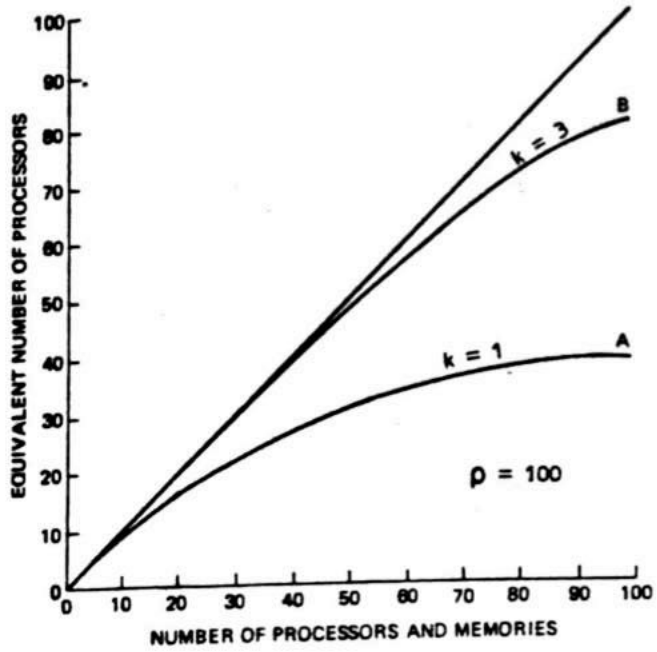

Fig. 12. Improving throughput in hypothetical systems

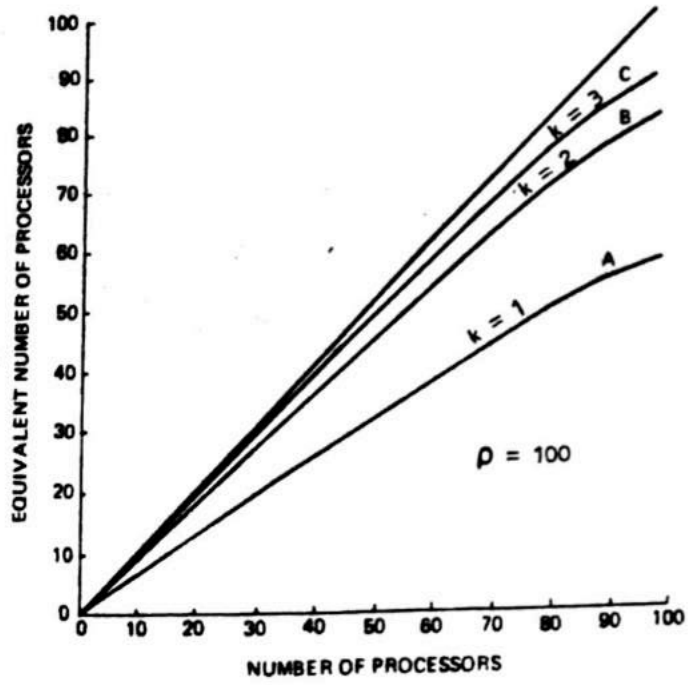

Fig. 13. Hypothetical systems with the same number of processors and memories

VII. REFERENCES

[1] Anastas, M. S. and Vaughan, R. F. Parallel transition machines. *Proc. of 1979 Int. Conf. on Parallel Processing.* Aug. 1979.

[2] Arulpragasam, J. A.; Giggi, R. A.; Lary, R. F.; Sullivan, D. T.; and Wu, C.-C. Modular minicomputers using microprocessors. *IEEE Trans. Comp.* Vol. C-29. No. 2. Feb. 1980, pp. 149-160.

[3] Asher, J. R. and Skinner, C. E. Effects of storage contention on system performance. *IBM Sys. Jour.* No. 4. 1969, pp. 319-333.

[4] Baer, J. L. A survey of some theoretical aspects of multiprocessing. *ACM Comp. Surveys.* Vol. 5, No. 1. Mar. 1973, pp. 31-80.

[5] Baskett, F. and Smith, A. J. Interference in multiprocessor computer systems with interleaved memory. *Comm. ACM.* Vol. 19. No. 6. June 1976, pp. 327-334.

[6] Bhandarkar, D. P. Analysis of memory interference in multiprocessors. *IEEE Trans. Comp.* Vol. C-24. No. 9. Sept. 1975, pp. 897-908.

[7] Boyse, J. W. and Warn, D. R. A straightforward model for computer performance prediction. *ACM Comp. Surveys.* Vol. 7. No. 2. June 1975, pp. 73-93.

[8] Burnett, G. J. and Coffman, E. G. A combination problem related to interleaved memory systems. *Jour. ACM.* Vol. 20. No. 1. Jan. 1973, pp. 39-45.

[9] Buzen, J. P. and Denning, P. J. Measuring and calculating quene length distributions. *Computer.* Vol. 13. No. 4. April 1980, pp. 33-44.

[10] Chang, D. Y.; Kuck, D. J.; and Lawrie, D. H. On the effective bandwidth of parallel memories. *IEEE Trans. Comp.* Vol. C-26. No. 5. May 1977, pp. 480-490.

[11] Dijkstra, E. W. Solution of a problem in concurrent programming control. *Comm. ACM.* Vol. 8. No. 9. Sept. 1965, pp. 569.

[12] Enslow, P. H. Multiprocessor organization—a survey. *ACM Comp. Surveys.* Vol. 9. No. 1. Mar. 1977, pp. 103-129.

[13] Flores, I. Derivation of a waiting-time factor for a multiple-bank memory. *Jour. ACM.* Vol. 11. No. 3. July 1964, pp. 265-282.

[14] Habermann, A. N. Synchronization of communicating processes. *Comm. ACM.* Vol. 15. No. 3. Mar. 1972, pp. 171-176.

[15] Kuck, D. J. A Survey of parallel machine organization and programming. *ACM Comp. Surveys.* Vol. 9. No. 1. Mar. 1977. pp. 29-57.

[16] Madnick, S. E. Multi-processor software lockout. *Procs.—1968 ACM Nat. Conf.* pp. 19-24.

[17] Strecker, W. D. Analysis of the instruction execution rate in certain computer structures. PhD thesis. Carnegie Mellon Univ., Pittsburgh, PA, 1970.

[18] Vaughan, R. F. and Anastas, M. S. Microprocessor based transition machines. *Proc. of COMPCON FALL '79.* Sept. 1979.

[19] Vaughan, R. F. and Anastas, M. S. Extending a model of multiprocessor throughput performance. Submitted for publication, available from the authors.

[20] Wulf, W. and Bell, C. G. C.mmp—a multi-mini-processor. *1972 Fall Joint Comput. Conf. AFIPS Conf. Proc.* Vol. 41. Part II. Washington, D.C., Spartan. 1972, pp. 767-777.