LIMITING MULTIPROCESSOR PERFORMANCE ANALYSIS

Russell F. Vaughan and Mark S. Anastas
Boeing Aerospace Company
P.O. Box 3999
Seattle, Washington 98124

Abstract -- This paper describes an analysis of the major sources of overhead in multiprocessor systems with emphasis on performance equations for large systems. A model is developed for studying the relative contributions of these sources of overhead. The traditionally treated problem of memory contention is shown to be containable within bounds with limit equations provided. Software control table lockout on the other hand is shown to be beyond containment in large systems such that an upper limit on performance exists. Effective methods of reducing lockout overhead are explored. Control program efficiency is shown to be the only means of achieving very large multiprocessor systems which are efficient. It is shown also that if such efficiency could be obtained in a centralized control mechanism (by hardware or other means), there are no other immediate theoretical problems associated with increasing multiprocessor size.

Introduction

There are known limitations to single processor approaches to increasing general purpose computer throughput capabilities [8]; moreover, requirements for increased throughput seem more general and insatiable than ever. The advent of inexpensive microprocessors has emphasized the necessity for an effective multiprocessing technology capable of effectively 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. It is therefore the limiting performance behavior where many processors are involved that is the central theme of this paper.

The theoretical problems associated with deadlock avoidance and synchronizing concurrent processes have been solved. [3],[9],[13] The practical problems however, which are encountered when implementing large multiprocessing systems have seemed unavoidable. To address these practical issues, a general parameterized model of the major overhead contributions in multiprocessing systems is presented. Descriptions of the individual overhead contributions modeled separately are found in the literature, but not integrated mathematical models as presented here. Nor has the emphasis of these other models been on performance expectations in the limit as system size increases. The model described in this paper relates the three major contributions to overhead in multiprocessing systems 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 Figure 1. These are the following:

1. System Control. The multiprocessor executive control program execution time requirements.

2. Control Table Lockout. To provide coordinated control, common queues are required which imply critical sections in the control program which accesses 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.

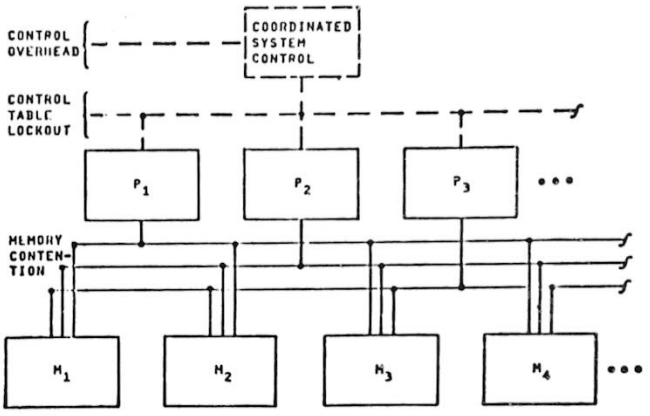


FIGURE 1: MODELED SOURCES OF OVERHEAD

To obtain a comprehensive model of multiprocessing overhead, without inappropriate complexity, a hierarchical model has been developed. The levels and states in this hierachy are the obvious ones. Figure 2 is a state diagram of the time expenditure states at the top level in this model of a multiprocessor system. 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 of current multiprocessing. (One of the

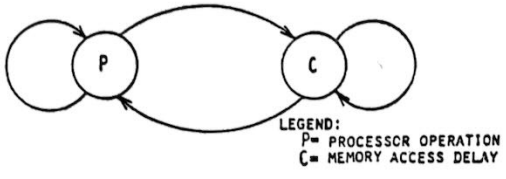


FIGURE 2: TIME EXPENDITURE STATES

design trades considered further on investigates potential advantages resulting from changing the temporal distribution.)

Time overhead (throughput) is the multiprocessing concern here. Other aspects of multiprocessing including memory and peripheral sizing have been modeled in reference [6]. These other aspects are very important in a system, and should be optimized to obtain the best performance for any given configuration. But they are not the major obstacles to a viable multiprocessing capability.

Processor Time Expenditure Model

The time utilization characteristics of the various activities that can be assigned to the processor are modeled here. In a multiprocessor system, it is expected that for some of these activities the amount of time expended may be dependent upon 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 Figure 2 can be modeled in more detail as shown in the state diagram of Figure 3. The four

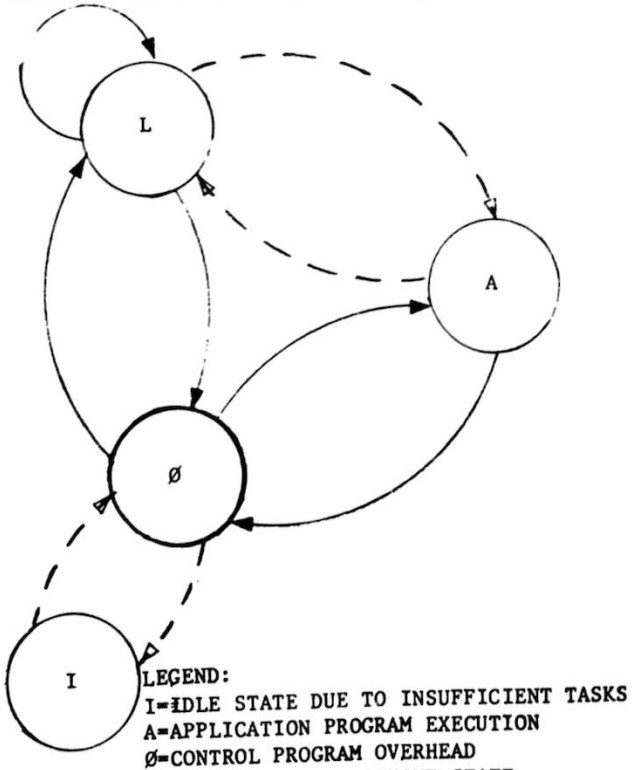


FIGURE 3: SECOND LEVEL: TIME EXPENDITURE SUBSTATES WITHIN THE PROCESSOR TIME EXPENDITURE STATE, P

states in this diagram are the following:

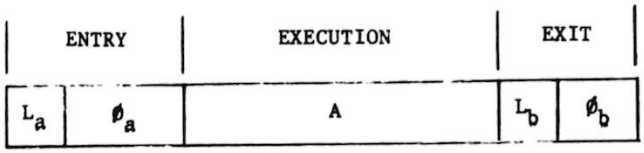
1. Idle state, awaiting an eligible application program task,

2. Application task execution,

3. Control program execution, and

4. Control table lockout.

In order to get performance predictions independent of the software configuration, it has been assumed that the idle state will be null. We are only interested here in performance degradation not attributable to insufficient jobs to go around. (Utilization considerations will be discussed later on however.) It is also assumed that lockout will only be experienced 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 timeline in Figure 4 shows the phasing among the remaining three states.



A = APPLICATION PROGRAM EXECUTION TIME

$\emptyset = \emptyset_a + \emptyset_b$ = MULTIPROCESSING EXECUTIVE OVERHEAD

$L = L_a + L_b$ = LOCKOUT OVERHEAD FOR ACCESSING CONTROL INFORMATION COMMON TO MULTIPLE PROCESSORS

FIGURE 4: TASK TIMELINE BASIS FOR PROCESSOR OVERHEAD

Each of the three remaining processor time expenditures is modeled very simply in the following. 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 appreciably in these models, but it has been found that performance predictions are relatively insensitive to such improvements. The simpler models are easier to describe and understand, and fit existing multiprocessor performance data very adequately.

Application Program Task Execution

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

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, ϕ_j.

$$\phi \equiv \sum_{j=1}^{J} \phi_j = J\phi_j$$

The control program is assumed to require the same constant total amount of execution time, ϕ, for each task. It is also assumed that its execution time is independent of the number of processors in the system. The latter of these assumptions supposes that queues are implemented with multiple pointers such that the lengths of queues do not result in a commensurable amount of searching to process linked task lists. This seems to be a unilateral approach to sophisticated control programs appropriate to multiprocessing.

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. It is assumed that 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 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 $L_j = N_j\phi_j = N_j \frac{\phi}{J}$. 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. 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.

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

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

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

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, we obtain:

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

The expected number of locked out processors, N_1 is plotted in Figure 5 for various values of ρ.

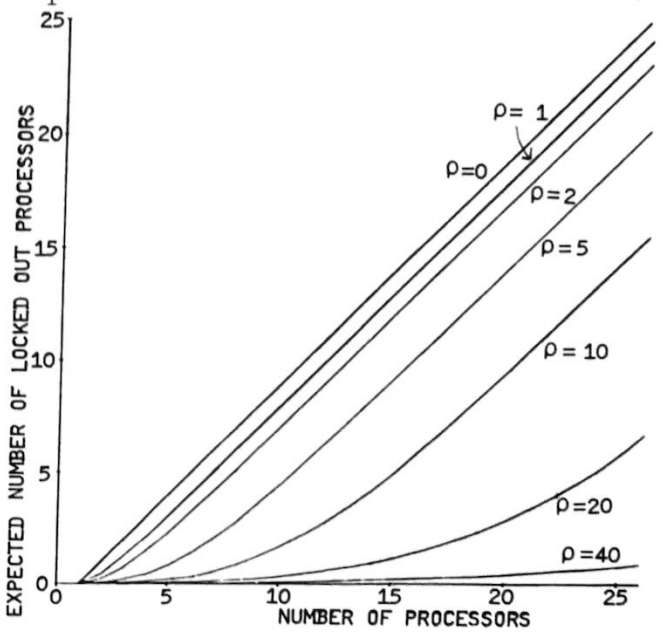


FIGURE 5: IMPACT OF PROCESSOR EFFICIENCY ON LOCKOUT OVERHEAD

These curves are in agreement with Madnick [15] in spite of a very different derivation. The significance of increasing the effective partitions in the control program will be discussed further on.

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, ϕ and L states. Equivalently we are interested in knowing the total number of processors in the configuration occupied by each activity. This assessment can be obtained by establishing the ratios of time spent in each activity to the unit of a processor's time. By defining X_A, X_ϕ, and X_L as the respective ratios for the A, ϕ, and L activity states, it can be seen that:

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

Furthermore, the equivalent number of processors involved in each activity per unit time N_A, N_ϕ, and N_L can be determined as:

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

In order to establish these relative contributions, we can substitute in the results obtained previously. The unit of processor time for J=1 is thus seen to be:

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

$$N_A = \rho \cdot N \cdot U^{-1}$$

$$N_O = N \cdot U^{-1}$$

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

Memory Contention Delay Model

There are various memory/processor interconnection schemes that can be employed for access arbitration including multiport controllers and crossbar switches as described by Enslow [10] which effect the logical interconnecting paths shown in Figure 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 reference [17]. 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, we are concerned with the percentage of time that the processors spend waiting for a memory to service their requests [2],[12].

General Model of Synchronous Interleaved Memory

To simplify the model we have assumed equal likelihood of a processor accessing any of the memories on a given request. Address interleaving makes that a realistic assumption. In addition, it has been assumed that each processor synchronously makes a memory access each cycle; this is a worst case situation tending to make the resulting performance predictions pessimistic rather than optimistically unrealistic.

We will begin by defining the probability, $P_S(i)$ of exactly i processors converging on single memories anywhere in the system on a given access:

$$P_S(i) = \sum_{j=1}^{\lfloor \frac{N}{i} \rfloor} P_S(i,j), \text{ where } \lfloor x \rfloor \text{ is the largest}$$

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. (For a detailed treatment of probability theory, refer to Feller [11].) To proceed, we will 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 if there are j instances of such convergence in the system. Under the random accessing and equivalence between processors assumptions that we have made:

$p_p(i,j)=i\cdot\frac{j}{N}$, since ixj of the N processors are involved.

$p_m(i,j)=\frac{j}{M}$, 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:

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

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

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

Modeling Memory Response Time

So far we have only been dealing with the probabilities of processor/memory convergence, whereas what we are really interested in is contention situations where processor time is lost. We 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 the ratio of processor request time over memory response time. A new conditional probability, $P_R(i)$ can therefore be defined which is the probability that a processor involved in an i-way convergence situation will actually experience contention:

$$P_R(i) = \frac{(i-k)}{i}, \text{ for } i>k; \; P_R(i) = 0, \text{ otherwise.}$$

Then the probability a processor will experience memory contention due to i-way convergency situations is:

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

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

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

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

$$P_C = \sum_{i=k+1}^{N} P_C(i), \text{ since contention can only occur}$$

when i>k. Therefore we have:

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

Approximating the Distribution Function

We are left then with the requirement for obtaining a distribution function $P_m(i)$. Many such models of processor queueing on individual memories have been advanced [2],[7]. It has been shown that 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 shown to be small, since we are primarily interested in configurations for which $M \geq N$ and $k \geq 1$. Bhandarkar [5] has shown percentage errors of less than 5 percent in all cases for the model assumed here.

The model that we have selected is the binomial approximation of Strecker [16] which was found to "work well in all cases" by Baskett and Smith [4] and with more accuracy for $M \geq N$ by Bhandarkar [5]. 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 on a given cycle is:

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

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 Figure 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.

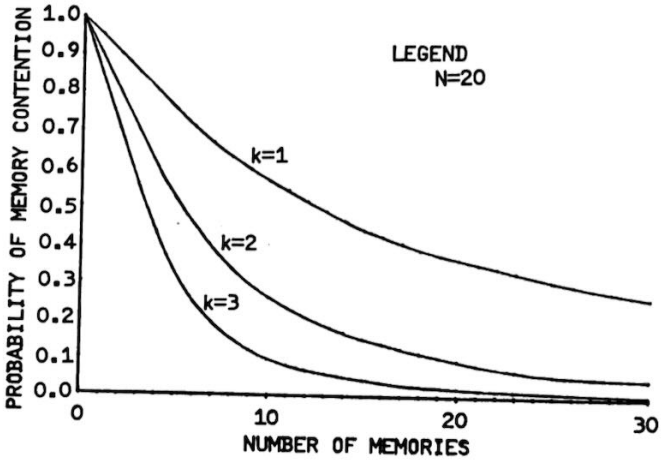


FIGURE 6: IMPACT OF RATIO OF PROCESSOR REQUEST TO MEMORY RESPONSE TIMES

It should be noted that 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., $P_m(i) = P_m(i, M, N)$.

Limiting Behavior of "Square" Systems

It is interesting to 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. We will refer to systems for which M=N as "square" multiprocessors, and define the notation: $P_c(M=N,\emptyset,k) = P_c(M,M,k) = P_c(N,N,k)$. 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 noticing that $\sum_{i=0}^{N} P_m(i,M,N)=1$, we obtain:

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

To obtain a limit for P_c, we have substituted M=N into $P_m(i,M,N)$ and used the limit

$$1/e = \underset{N\to\infty}{\text{Limit}} \left(1-\frac{1}{N}\right)^N.$$

Then for "square" systems:

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

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

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

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 characteristics which are responsible.

1. Processor operations do not typically require an access on every cycle of the instruction. Statistically, somewhat less than half of the TI 9900 microprocessor machine cycles require a memory access, for example.

2. Some processors implement a cache memory scheme for look-ahead memory accessing to reduce the average wait time in the processor. This reduces the number of cycles for which the processor makes memory accesses, but substantially increases the number of accesses outstanding when they are made.

These (in general combined) phenomena establish an effective, although 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, and approximately constant access duty cycle, d can be expected which 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 .N'. Real "square" systems would then be characterized by the model as "rectangular" systems of dimensions N= η.M, where η is the apparent asymmetry ratio.

Limiting Behavior of "Rectangular" Systems

It is interesting to consider memory contention effects when system size is increased in congruent rectangular form. Just as was the case for "square" systems, it can be seen that for large "rectangular" systems the contention probabilities level off to approximately constant values. Chang, Kuck and Lawrie [8] derived an expression for the limit from the memory's viewpoint (the probabilitiy 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 using the same approach as described previously for "square" systems.

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

Accuracy considerations relying on Bhandarkar's [4] data suggest $\eta \leq 1$ as the primary domain of usefulness for this equation. The limits for k=1 and for asymmetry values η =1, 1/2 and 1/3 are shown in Table I. The asymptotic approach to these limits is shown in Figure 7.

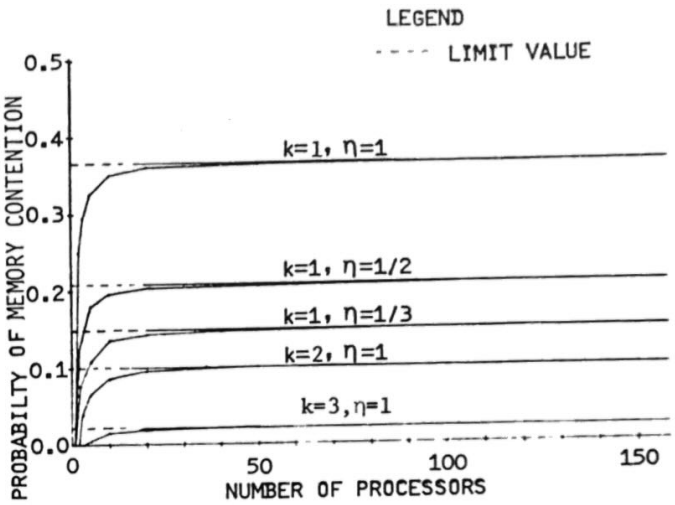


FIGURE 7: ASYMPTOTIC APPROACH TO LIMITING MEMORY CONTENTION PROBABILITY VALUES

Combining Processor and Memory Contention Overhead

In the previous accounting of processor time expenditures, there were only three categories corresponding to the three processor states of application program, control program and control table lockout. It must now be acknowledged that 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, we have: $C=(A+\emptyset+L)\cdot P_c$. Thus, if we define the respective primed quantities to represent the time in each state exclusive of memory contention, we have: $A+\emptyset+L=A'+\emptyset'+L'+C$ and therefore: $A'+\emptyset'+L'=(A+\emptyset+L)(1-P_c)$ and the respective number of 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. 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 Figure 8. The dashed line represents the extrapolation from data presented by Bhandarkar [5] which resulted from a more accurate Markov chain model for k=1, η =1.

The form of the other three expected numbers of processors N'_A, $N'_\emptyset$ and N'_L can be obtained by substitution from previously obtained solutions for N_A, $N_\emptyset$, N_L and P_C. It should be clear that

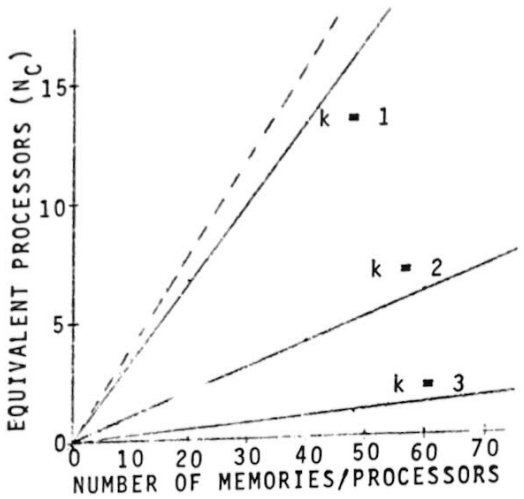


FIGURE 8: N_C FOR "SQUARE" SYSTEMS

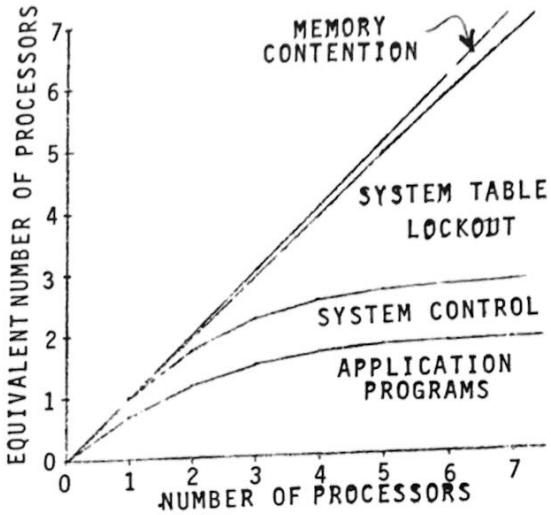


FIGURE 9: OVERHEAD CONTRIBUTIONS FOR $\rho=2, k=1, \eta=1, M=50$

N_A' provides a desirable measure of throughput in multiprocessors. It provides the effective number of processors being applied to the application programs.

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:

$$\tau \equiv \lim_{N \to \infty} N_A' = \rho \cdot (1-P_C)$$

The trailing factor may approach a limit as well, since in general P_c is a function of N. Thus, the control program efficiency not only determines the utilization per processor, but also the maximum achievable throughput of the entire machine. In Figure 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 previous 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 ρ_M of the memory accesses:

$$\tau = \rho_p \cdot \rho_M.$$

Design Trades in Multiprocessors

It is significant that in the example shown in Figure 9, memory contention is not responsible for the reduced efficiency of processors as a function of their increased number. This is not to say that memory contention cannot be a very significant overhead factor, but rather that it is a problem which has been solved by the existing multiprocessing technology. In the example, memory contention is reduced to insignificance by the large number of memory modules (M>>N). Another method which 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 it is not a stringent requirement to obtain relatively fast memories.

Reducing Memory Contention

There are of course many configurations for which memory contention appears to be very significant. In the solid lines in Figure 10, the situation previously presented in Figure 9 has been modified to include only 5 rather than 50 memory modules. In this example, there is actu-

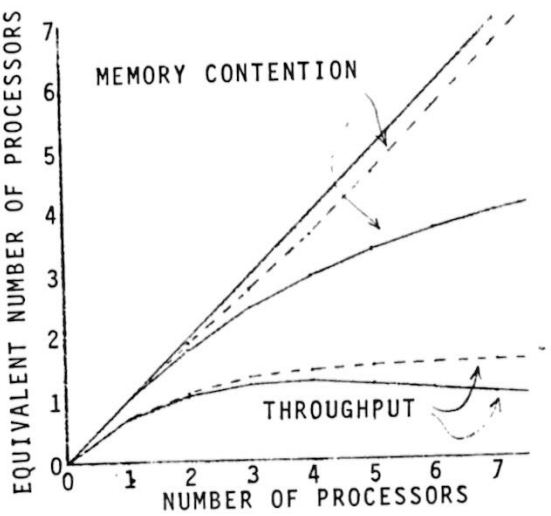


FIGURE 10: REDUCING MEMORY CONTENTION

ally a negative improvement in application program throughput for more than 4 processors. The reason for this negative return can be seen to be attributable to the increasing number of locked out processors. 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 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. By estimating P_C for 5 memories and N_A+N_0 processors, we obtain the revised overhead plots shown as dotted lines. The marginal gain in performance for few processors can be seen. 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 [12] for a similar conclusion. Memory contention is not the peril of multiprocessing.

Reducing Processor Lockout

It is clear from the preceding discussion that lockout is the primary contributor to multiprocessor inefficiency for large numbers of processors. Let us therefore consider various means by which it can be reduced. The starting point of course is the consideration of 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:

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

2. Reduce the execution time of the critical sections in the control program, and

3. Partition the control program into many separate rather than a single common critical section.

The relative effectiveness of the various methods of reducing lockout overhead ultimately depends upon the design of the control program itself. There are upper limits for each of these 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.

It should be apparent that these three solutions have direct analogs in the reduction of memory contention which 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, with the improvements all being relative to the system whose performance characteristics were shown in Figure 9. Line A in Figure 11 represents this baseline system's throughput performance.

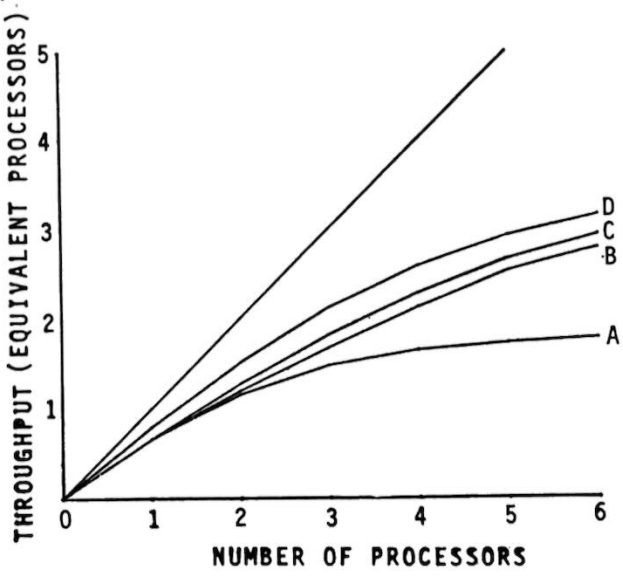


FIGURE 11: COMPARISON OF APPROACHES TO REDUCING CONTROL TABLE LOCKOUT

Limiting Control Program Lockout. It is not actually necessary for the entire control program to be locked out such that only one processor can be executing it at any one time. In Figure 11, line B, the expected performance is shown for a system whose control program need be locked out only half of the time. The data for the figure was 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 non-locked-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 Figure 11, line B, $\rho=2$, Z=0.5, and therefore $\rho_{LOCKOUT} = 5$.

Control Program Efficiency. An obviously effective method of improving multiprocessor throughput is by 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 $\rho=2$.

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 assuming there were J partitions of the control program with independent critical sections. This equation was used to obtain the performance indicated in Figure 11, line C, for J=2. In reference [18] it was also suggested that a small number (2, 3 or 4) or partitions significantly improve efficiency. It should be obvious that the limiting number of partitions that could be incorporated is not a large number however.

Increasing Individual Processor Efficiency.

The level at which application programs interface with the control program has the same impact on efficiency as does the overhead involved in the control program. If the execution time of the typical application program task is increased such that the number of executable instructions is doubled, the same efficiency advantages 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 article by assuming that there are no processors in the idle state. (See Figure 3.) The job control languages of batch processing systems largely determine the task level. This is a critical issue particularly in mainfram multiprocessing, but one which is beyond the scope of the current article.

Kuck [14] investigated the potential for breaking up general problems into parallel segments to obtain commensurable speedup. The inherent parallelism was shown to be roughly proportional to the size of the application program, if the program units which are dispatched are taken to a low enough level. This is in contrast to what was formerly thought to have been an order of log relationship [14]. Thus, there is potential in the programs themselves for solution by parallel arrays of slow processors to obtain very high throughput. But this level would reduce the effective value of A by orders of magnitude which in turn reduces ρ (and with it feasibility) by orders of magnitude. And thus, methods which artifically increase ρ do not attack the multimicroprocessor program.

The Future of Multiprocessing

It has been demonstrated that 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.

Let us consider the potential of such solutions to determine whether there are other theoretical problems. Figure 12, line A illustrates the system described originally in Figure 9, but assuming 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 seen to be about 40 processors. But as shown in Figure 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.

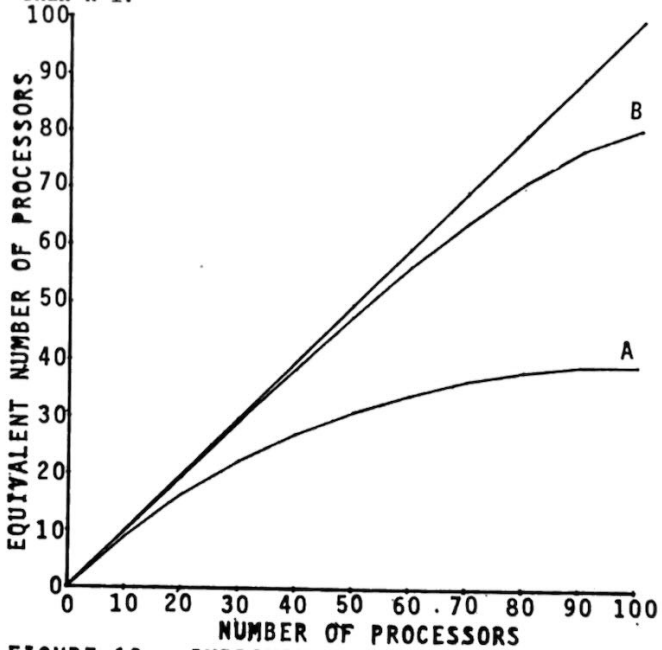


FIGURE 12: IMPROVING THROUGHPUT IN HYPOTHETICAL SYSTEMS

In going to such high throughput systems, however, there would be requirements for commensurably larger numbers of memory modules. Figure 13, line A, illustrates the situation for ρ =100 with "square" multiprocessors (M=N) and k=1. The

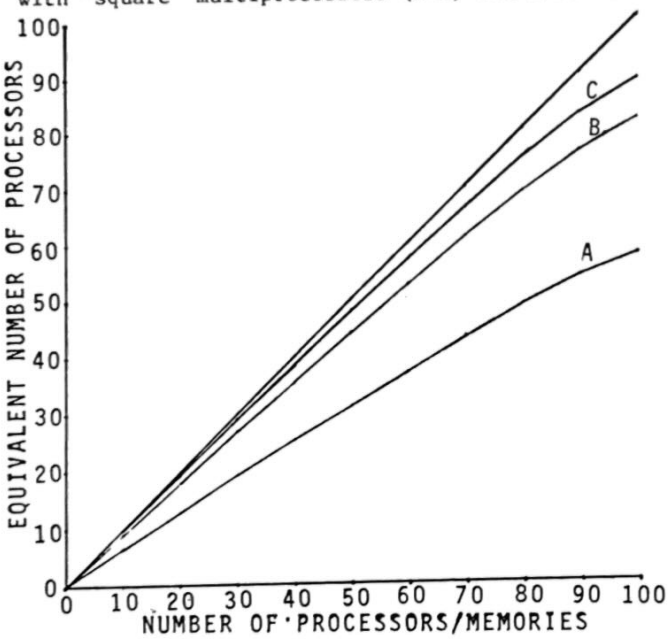


FIGURE 13: HYPOTHETICAL SYSTEMS WITH THE SAME NUMBER OF PROCESSORS AND MEMORIES

improvement in contention with increasing memory response time can be seen in lines B and C respectively for k=2 and k=3.

Conclusions

It has been shown that 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 increasing this parameter. Exploring methods of increasing hardware support for the control programs is therefore the most likely avenue to extending the limits for multiprocessor throughput performance. A companion paper discusses such an approach for which multiprocessor control can be made extremely efficient [1].

References

[1] Anastas, M. S. and Vaughan, R. F., "Parallel Transition Machines", Proc. of 1979 Int. Conf. on Parallel Processing (Aug. 1979)

[2] Asher, J. R. and Skinner, C. E., "Effects of Storage Contention on System Performance", IBM Sys. Jour., No. 4 (1969) pp 319-333

[3] Baer, J. L., "A Survey of Some Theoretical Aspects of Multiprocessing", ACM Comp. Surveys, Vol. 5, No. 1 (Mar. 1973) pp 31-80

[4] 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

[5] Bhandarkar, D. P., "Analysis of Memory Interference in Multiprocessors", IEEE Trans. Comp., Vol. C-24, No. 9 (Sept. 1975) pp 897-908

[6] 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

[7] 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

[8] 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

[9] Dijkstra, E. W., "Solution of a Problem in Concurrent Programming Control", Comm. ACM, Vol. 8, No. 9 (Sept. 1965) pp 569

[10] Enslow, P. H., "Multiprocessor Organization - A Survey", ACM Comp. Surveys, Vol. 9, No. 1 (Mar. 1977) pp 103-129

[11] Feller, W., An Introduction to Probability Theory and Its Applications, Vol. I, Wiley, New York (1968)

[12] Flores, I., "Derivation of a Waiting-Time Factor for a Multiple-Bank Memory", Jour. ACM, Vol. 11, No. 3 (July 1964) pp 265-282

[13] Habermann, A. N., "Synchronization of Communicating Processes", Comm. ACM, Vol. 15, No. 3 (Mar. 1972) pp 171-176

[14] Kuck, D. J., "A Survey of Parallel Machine Organization and Programming", ACM Comp. Surveys, Vol. 9, No. 1 (Mar. 1977) pp 29-57

[15] Madnick, S. E., "Multi-Processor Software Lockout", Procs. - 1968 ACM Nat. Conf., pp 19-24

[16] Strecker, W. D., "Analysis of the Instruction Execution Rate in Certain Computer Structures", PhD thesis, Carnegie Mellon Univ., Pittsburgh, PA, 1970

[17] Vaughan, R. F. and Anastas, M. S., "Microprocessor Based Transition Machines", Proc. of COMPCON FALL '79 (Sept. 1979)

[18] 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, DC, Spartan, 1972, pp 767-777