Performance evaluation of the MP/C

by BRUCE W. ARDEN
Princeton University
Princeton, New Jersey

and

RAN GINOSAR
Bell Laboratories
Murray Hill, New Jersey

ABSTRACT

The performance of the MP/C, a multiprocessor/multicomputer architecture, is evaluated by means of deterministic and probabilistic techniques and simulations. The MP/C is examined in two different applications, as a tree machine and as a multiprogramming general-purpose system. The second application of the MP/C is compared with a star architecture. The evaluation suggests that the MP/C is very effective for certain special applications, such as tree algorithms, and worst-case analysis shows that its performance is acceptable even for general purpose computations.
A. INTRODUCTION

The architecture of the dynamically reconfigurable MP/C has been described in detail. It consists of a linear bus with multiple processors and memory modules. Figure 1 shows a simplified MP/C system and a few possible configurations. Switches are positioned on the bus between every pair of processors, and the bus can be partitioned into any configuration of segments. On each such segment, only one processor is active, and it accesses all the memory modules on its segment. Open switches provide effective isolation of segments and guarantee full mutual exclusion of active processors. A processor may activate another processor in its segment by opening the switch immediately to the left of the activated processor. The memory space of the activator is split, and part of it is exclusively assigned to the activated processor. The switch control logic supervises this reconfiguration operation. Such an activation is, in effect, a hardware implementation of a process-fork operation. The converse operation, process-join, is achieved by deactivating a processor, closing the switch to its left, and thus reattaching its memory to the neighboring active processor.

The applicability of most indirect performance evaluation tools, such as mathematical analysis and simulation, to an architecture as complex as the MP/C is rather limited. The many unconventional attributes of the MP/C, especially its reconfigurability, are beyond the capabilities of most analytic techniques. Simulations also involve a lot of approximations, which reduce the usefulness of their results. The desirable techniques would be an emulation system (which inputs, executes and measures real MP/C code) and a measurement on a full-scale hardware implementation. Both techniques are currently being pursued.

In this report we describe some preliminary performance evaluation studies. Two situations are considered. In the first we examine the MP/C in the context of tree-structured computations. As is emphasized elsewhere, it is expected that this application is the most appropriate one for the MP/C. The second part investigates the performance of the MP/C in running a general-purpose, multiprogramming operating system.

Figure 1—Simplified diagram of an MP/C with possible reconfigurations
and many independent tasks. Although this is not the best application for the MP/C, the evaluation serves two purposes. First, not every algorithm is tree structured, and it is economical to have a machine that is efficiently applicable to more than one computational structure (preferably, it should provide all the computational needs of its owner relatively efficiently). Second, it is a worst-case study: given an unfavorable computational structure, what is the lower bound on the performance of the MP/C?

The next section analyzes the performance of tree-structured algorithms. Section C introduces multiprogramming applications on the MP/C. Section D employs a deterministic model to arrive at some preliminary conclusions. Sections E through I describe the analytic models and their results. We focus on techniques for modeling preemption, explain the MP/C and star models, present the results and their interpretation, and describe an alternative MP/C model. The simulation study is described in Sections J through L.

B. TREE-STRUCTURED ALGORITHMS

Since only one processor is active in each MP/C partition, the throughput (in terms of the collective instruction execution rate of all the processors) varies, and depends on the number (or, equivalently, size) of the partitions. Since the partitioning is really an image of the process structure of the running algorithm, the amount of parallelism in the MP/C generally reflects closely the level of inherent parallelism in the algorithms. For example, consider a divide and conquer algorithm.

Five phases can be identified:

1. The input phase, in which the data and programs are loaded into the MP/C. This phase probably takes close to the same amount of time on the MP/C or any other organization, so it can be dropped from consideration.
2. The forking phase, in which a series of activations follows top-down scanning of a binary tree. In each step the number of active processors is doubled. In the merge-sort example, there is only a limited amount of computation in each step, provided the data and program are loaded effectively. In that case, this phase takes $O \left( \log n \right)$ time, with $n$ processors. In some algorithms, like parallel search, all leaf processes are spawned by a single parent in the process-forking tree, rather than recursively spawned according to a binary tree. In those cases, the successive activation may be replaced with the single step PARALLEL-activate operation, in which all processors are activated simultaneously.
3. In the totally parallel phase all processors are active, and throughput is maximal. When the number of data elements involved is larger than the number of processors (usually the case), each processor executes a sequential algorithm on its subset of the data during this phase. In the merge-sort example with $N$ data elements and $n$ processors this phase takes $O \left( \left( \frac{N}{n} \right) \log \left( \frac{N}{n} \right) \right)$, which is close to a linear speedup (of this phase only). Similar speedup is observed for other examples.
4. The joining phase, which is the converse of phase 2. In merge-sort, at each successive step the merge operation runs twice as long while employing only half the number of processors, and takes time $O \left( N \right)$, usually longer than the previous phase. On the other hand, search and retrieval algorithms do not require heavy computation at this phase (unless a substantial amount of data is retrieved), and some other algorithms require only $O \left( \log n \right)$ time.
5. The output phase, which has a structure similar to the input.

It is clear that maximum throughput (and linear speedup) is achieved only in phase 3. Thus, the throughput depends on the ratio of the time complexity of phase 3 to that of phases 2 and 4. Similar behavior is demonstrated by other classes of algorithms.

Most algorithms execute on the MP/C with asymptotic time complexities similar to what they would have on other parallel-processing architectures. However, constant factors may differ greatly, and indeed might make all the difference. The most important advantage of the MP/C is that regard is its ability to switch memory blocks. Where multicomputers share data by moving them around in messages, the MP/C avoids most such data movements. For applications like merge-sort, where all the data have to be moved in a tree multicomputer, this moving takes $O \left( N \right)$ time for $N$ data elements. In addition, the overhead associated with message passing (packing, serializing, unpacking, etc.) is saved. On the other hand, shared-memory multiprocessors also avoid message passing, but the contention associated with the shared resources limits the effective number of processors, as we have said.

In summary, the performance advantage of the MP/C is characterized both by an improvement of constant factors and by an ability to support many processors. However, it is difficult to prove constant-factor improvement without actually implementing the proposed architecture in hardware and comparing its measured performance to that of a hardware implementation of other architectures.

C. THE MP/C AS A GENERAL-PURPOSE, MULTIPROGRAMMING COMPUTER

In considering the utility of the MP/C for running a general-purpose operating system, and its associated independent user tasks, a different approach is taken. As long as all processors run user tasks, and as long as those tasks require only computing, the throughput is maximal. When a task, running on $P_i$, requires system services, or when it completes and has to be swapped, $P_0$ has to access $M_i$ by preempting all the intermediate processors. We model the effect of preemption through the multiclass network of queues. Each running process is characterized by the ratio of pure computing time to I/O time it requires, and this ratio can vary from one process to another.

Utilization of memory is also an important performance factor, especially in a microprocessor-based MP/C where a single memory block may be more expensive than its associ-
ated processor. Although it seems that it would be possible to effectively employ more processors on the same amount of memory in the MP/C than in strict multiprocessors, the fact that common programs cannot be shared and each running processor must store its own copy of the program reduces the potential memory utilization. This effect, however, should be weighed against the relative benefits of the MP/C.

Three methods were employed: deterministic evaluation, analytic modeling using queueing networks, and simulation. The purpose of the evaluation study was to find lower bounds on the performance. That is, we model the MP/C under the least favorable conditions, and try to evaluate its performance.

Since absolute performance figures are generally not meaningful, this is mainly a comparative evaluation. We first identify an unique feature of the MP/C (certain preemptions), and compare it to another architecture (the star) that differs from the MP/C only in that feature. Thus, the comparison isolates the effects of that feature.

There is no single architecture which can efficiently execute the full range of concurrent computations, but the reconfigurable MP/C is intended to be usable over a fairly wide spectrum. The hierarchical nature of the MP/C makes it a candidate for the execution of a general-purpose, process-structured operating system (e.g., UNIX). In such an environment, user processes and most of the system processes are spawned by the kernel process, and constitute a shallow but wide tree. That is, most processes are direct children of the kernel. On the MP/C, such a system would run the kernel on Po, and the user processes on the other processors. For simplicity, assume that each user is allocated a single processor and a single memory module, and that no inter-user communications are needed. Also, we ignore here the possibilities of running a user process on Po (in addition to the kernel), of running the kernel on another processor, and of using a distributed kernel.

It is known that the speedup of such a multiple-processor organization is generally sublinear. Among the factors that usually contribute to that are memory contention, system contention, and synchronization between processes.  

In addition, another source of degradation is unique to the MP/C. Whenever a process that runs on some processor Pi has to communicate with the kernel, the bus segment starting at the kernel processor and up to Pi must be available, that is, all the processors between the Pi, 0 < j ≤ i must be preempted. The only active processor in this segment would be Po, running the kernel. Such communications are required when a process initiates I/O operations, when it receives resulting data, when a page fault occurs, when the process initiates any type of system service request, and when a process terminates and the next process has to be loaded. The evaluation of the effect of this requirement is the subject of this paper.

D. DETERMINISTIC MODELING

Consider a deterministic model for the star. The kernel executes on Po, and the Pi, i = 1,...,N are user processors. Each user processor is allocated a single user job, which goes through the following fixed cycle. It first executes on the user processor for β units of time, and then requires service from the kernel for one unit of time. Since there are N users, and each requires one time unit of service by the kernel, the cycle time of each job is

$$C_{\text{star}} = \max \{\beta + 1, N\}. $$

When N < β + 1, the kernel processor Po is underutilized. When N > β + 1, interference between jobs (congestion at the kernel) increases cycle times beyond the necessary minimum β + 1. Hence, we define a balanced star system as one with N = [β + 1]. Note that β + 1 can also be interpreted as effective parallelism. Given N, for β < 1 the kernel is highly congested and the whole system behaves effectively as a uniprocessor. On the other hand, for β > N the system shows linear speedup without any interprocessor interference. The utilization of the kernel processor is

$$P_{0, \text{star}} = N \left[ \frac{N}{N + \beta + 1} \right] = 1.$$  

The utilization of each user processor is

$$P_{i, \text{star}} = \left( \frac{\beta}{\beta + 1} \right)_{N \sigma / i} = \frac{\beta}{\beta + 1}, i = 1,...,N.$$  

In Figure 2 we show an example of a Gantt chart [C1, C4] of a balanced system with N = 3 and β = 2. Since the system is balanced, there is no queuing at the kernel, and the kernel is fully utilized. This is a snapshot of 6 time units during steady state operation. The tasks at Po are labeled by the user processor which they serve. The slanted areas designate idle periods at the user processors.

Now consider a similar model for the MP/C. The difference is that when the kernel serves the job of Pi, all user processors between Pi and the kernel must be preempted. This is represented by a task that executes concurrently on all of them. In Figure 3, such tasks appear as the vertical columns. Note that only the topmost processor in a vertical column is considered active for the purpose of computing utilization. As before, each job requires 1 time unit of service from the kernel, and
Figure 3—Deterministic scheduling, MP/C with fixed \( \beta \)

\( \beta \) units of computation on the user processor. The kernel processor utilization is

\[
\rho_{0,\text{MP/C}} = \frac{N}{(\beta + N)},
\]

and the utilization of each user processor is

\[
\rho_{i,\text{MP/C}} = \frac{\beta}{(\beta + N)}, \quad i = 1, \ldots, N.
\]

Unlike the star, MP/C has no value of \( N \) for which any processor is fully utilized, and interference always exists. Instead, we say that the MP/C is a balanced system when overall utilization is maximized. One obvious case is when \( \beta = \infty \), with \( N \) fixed. The other case is derived as follows. Given \( \beta \), and treating \( N \) as a continuous variable,

\[
\lim_{N \to \infty} \frac{\partial}{\partial N} \rho_{\text{MP/C}} = \lim_{N \to \infty} \left[ \frac{\beta}{(N + 1)} \left( \frac{\rho_{0,\text{MP/C}} + N \rho_{i,\text{MP/C}}}{N + 1} \right) \right] = 0.
\]

Also, for \( N = 0 \), \( \rho_{\text{MP/C}} = 0 \). Assuming \( \rho_{\text{MP/C}} > 0 \) in the open interval \( N \in (0, \infty) \), there exists a maximum for \( \rho_{\text{MP/C}} \) in that interval. We equate to zero the first derivative of \( \rho_{\text{MP/C}} \) with respect to \( N \):

\[
\frac{\partial}{\partial N} \rho_{\text{MP/C}} = \frac{\partial}{\partial N} \left[ \frac{N(1 + \beta)}{(N + 1)(\beta + N)} \right] = 0.
\]

We get

\[
N^2 + N(1 + \beta) + \beta = 2N^2 + N(1 + \beta)
\]

or

\[
N = \left\lfloor \sqrt{\beta} \right\rfloor.
\]

Since there is only one value of \( N \) in the interval \( (0, \infty) \) for which the first derivative is zero, and since \( \rho_{\text{MP/C}} \) achieves its minimum of zero in both ends, this is a maximum, and we need not consider the second derivative.

In other words, under these conditions the MP/C seems to support effectively less parallelism than the star. This relative advantage of the star is achieved at the cost of more elaborate (and more expensive) communication paths.

The asymmetry in the MP/C suggests that the MP/C can be better utilized if the load is not the same for all processors. In Figure 4, we take advantage of some of the idle time of the user processors, by introducing nonuniform loads of \( \beta_1, \beta_2, \beta_3 \) to \( P_1, P_2, \) and \( P_3 \) respectively. The kernel utilization is the same as above. The total (or average) utilization of the user processors is

\[
\bar{\rho}_{\text{Users}, \text{MP/C}} = \frac{1}{N} \sum_{i=1}^{N} \frac{\beta + i - 1}{\beta + N} = \frac{2\beta + N - 1}{2(\beta + N)}.
\]

Using the same procedure as above, we get

\[
N = 1 + \left\lfloor \sqrt{2(\beta + 1)} \right\rfloor
\]

as the value of \( N \) maximizes utilization. Although still lower than \( N = \beta + 1 \) of a balanced star, it is better than \( N = \left\lfloor \sqrt{\beta} \right\rfloor \) of the uniform-workload MP/C. In any case, the observation earlier made about effective parallelism is applicable here as well; given \( N \), low values of \( \beta \) imply effectively sequential processing, whereas \( \beta >> N \) assures virtually interference-free operation. The lesson of this discussion is that it is meaningful and practical to consider the MP/C only under high \( \beta \) workload.

E. ANALYTIC MODELING:
MODELING PREEMPTION

We present an approximate modeling of preemption that produces results for the steady-state analysis of preemption in a
network of queues. It is based on techniques employed for the modeling of message passing in multicomputer networks.¹ This technique may also be applicable to other queueing systems.

Assume that a job $j_1$, which must execute for $t_1$ units of time, starts execution at service center $c_1$. Job $j_2$ arrives at that center $t$ units of time later and preempts $j_1$ for time $t_2$, after which job $j_1$ resumes execution.

A variation of this situation, which is more interesting in the case of the MP/C, occurs when $j_1$ (running on $c_1$) is preempted for $t_2$ units of time, but $c_1$ is left idle and no other job executes on it during the preemption period $t_2$. This is the case when some resource that is allocated to $c_1$ is temporarily needed elsewhere (bus and/or memory in the case of MP/C). We model this idle preemption by a job that executes on the preempted processor for the same amount of time, $t_2$.

An accurate modeling of this sequence would be as follows. First, jobs $j_1$ and $j_2$ execute on centers $c_1$ and $c_2$, respectively. (See Figure 5a.) Then, $j_2$ spawns a new process, $j_2'$, which preempts $j_1$ and runs on $c_1$ (Figure 5b). Finally, $j_2$ releases $c_1$ and terminates, and $j_1$ resumes, as in Figure 5a again. However, the analytical tools do not allow the spawning of processes as part of internal transitions in queueing networks.⁷ Hence, another modeling technique is necessary.

In order to overcome this difficulty, we make use of the following observations. The details of scheduling are quite important for the outcome of the computation, but not all of the details are relevant for the steady-state performance analysis. In particular, it is important to know $t_1$ and $t_2$, but the relative order of execution, and the length of $\tau$ (the time from execution of $j_1$ and $j_2$ on $c_1$ as a sequence ($j_1$ executes first to completion and only then does $j_2$, execute), rather than having $j_2$ preempt $j_1$. (Note that in some cases this approach may not be valid.) Thus, the spawned job in the example above, $j_2'$, would not generally execute concurrently with its originator $j_2$, but rather at some later time, when $c_1$ can serve it. Since it is no longer mandatory that $j_1$ and $j_2$ execute simultaneously, the solution to the spawning problem is simple: when $j_2$ finishes execution on $c_2$, it changes its class from a productive one to a non productive, moves from $c_2$ to $c_1$, and executes there for time $t_2$, that is, the preemption time. This execution does not model real computation, but rather the time that $c_1$ is preempted and is not available for productive work. Following that, the job may return to $c_2$.

On the other hand, note that this technique also introduces two modeling errors. The fact that job $j_2$ has to spend $t_2$ in $c_1$ (in the non productive state) may delay the next activity of $j_2$ (for time $t_2$), and this might affect the total system performance, as it does in the case of the MP/C (see below). The second error is due to the successive possession of resources, rather than concurrent, as is implied by preemption. That is, in the model $j_2$ possesses $c_2$ and $c_1$ in sequence, rather than simultaneously. The waiting time to possess two resources concurrently may be longer than the sum of the waiting times to possess each independently. Thus, the first error may result in underestimate of the performance, whereas the second one probably has the opposite effect.

Among the available scheduling disciplines that can be modeled analytically, the last-come-first-served-preemptive-resume (LCFS-PR) discipline is the most suitable one.⁷ (Other disciplines, PS and in certain cases FCFS, may render the same results.)

Both productive processing and preemption time are thus modeled by active computation, although by different sets of classes. This separation of classes allows the evaluation of performance from two different viewpoints. For instance, the utilization in the productive classes reflects the real work done by the processor, for productive computation. On the other hand, the total utilization (productive and nonproductive) reflects that portion of time in which the processor is busy. Thus, if the productive utilization is 0.5 and the total utilization is 0.8, it can be concluded that although the processor is productively active only about half the time, no more than 0.2 additional utilization can ever be expected. Which of the two criteria is more interesting depends on the application.

F. ANALYTIC MODELING: THE MP/C MODEL

In the MP/C employed as a general-purpose computer, preemption is used to allow direct communications between the leftmost processor $P_0$ and some memory module $M_i$. All processors $P_j$, $0 < j < i$, must be preempted. The lost time is modeled by a job that starts at $P_{i-1}$ and progresses leftward until it runs on $P_0$. (See Figure 6). After finally finishing at $P_0$, this job returns to $P_i$ for the next cycle of computation.

All service times are drawn from exponential distributions. Two classes are associated with each processor $P_i$: the productive class $i$, in which a job runs on $P_i$, with expected service

![Figure 5](image)

![Figure 6](image)
time of one unit, and the nonproductive class designating this job when it goes through $P_{i-1}, \ldots, P_0$. The average service time on each of those processors is $\alpha$, the time it takes for $P_0$ to communicate with $M_i$. The transition probability matrix is thus trivial and contains only zero-one elements.

Note that an approximation error is introduced here. Suppose each single MP/C primitive operation (e.g., PREEMPT) takes $k$ units of time. In each $k$ units a processor can issue on PREEMPT instruction, which preempts only the next active processor down the line. Thus, the preemption of all processors between $P_0$ and $P_i$ is a multi-step operation too. Similarly, at each step the kernel can resume the operation of only one other processor. Hence, reactivating all the preempted processors is a multi-step operation too. In other words, instead of all these processes being preempted for $\alpha$ units of time, $P_i$ is preempted for $\alpha$ units, $P_{i-1}$ is preempted for $\alpha + 2k$ units, $\ldots, P_1$ is preempted for $\alpha + (i-1) \times 2k$ units of time, $0 < j \leq i$. However, as a first-order approximation, we assume that $2ki$ units of time are negligible compared to $\alpha$ units of time, and each processor is preempted for the same period. It may be possible to accommodate this detail later by adjusting the mean service times appropriately.

As explained above, many types of user-process-to-kernel communications may take place. In order to isolate the behavior of the MP/C from peculiarities of specific benchmarks and workloads, a simple execution profile was chosen. Each processor is assigned a single job. Each job does one of two things: it computes on its processor, or it requires service from the kernel. The first task is called computation, and the service time for it is a random variable drawn from an exponential distribution with mean of 1 unit. The second task is called I/O. It is exponentially distributed, and requires variable mean service time $\alpha$. On the MP/C, $\alpha$ represents the length of time for which the kernel has exclusive access to the memory of the current processor. High values of $\alpha$ represent I/O-bound jobs. By varying the ratio of the I/O time to computation time, $\alpha/1$, from 0 to $\infty$, we are able to study the performance of the MP/C across the full range of workload. For two reasons, this approach is superior to picking up some small set of workload values for which the MP/C performance is maximal. First, we have been able to uncover unexpected and rather surprising behavior at different workloads. Second, the MP/C is intended for general purpose applications, in which any type of workload can be expected. For instance, it is conceivable that jobs that have $\alpha$ values of 100 and 0.01 will execute on the same machine concurrently. Recall however that it was argued in Section D that the reasonable workload range is $\alpha < 1/N$ (corresponding to $\beta > N$).

At first, we model the case where all the jobs on the same MP/C have the same $\alpha$ value. Another case is when different jobs have different $\alpha$ values. It is reasonable to expect that, in the I/O phase, the farther the processor is away from $P_0$, the higher the incurred penalty for the whole system, because more processors must be preempted. Thus, it may be advantageous to assign the jobs with higher degrees of $\alpha$ to processors that are closer to $P_0$. We try to verify this assumption, and study the effect of optimal and nonoptimal assignment of processes.

G. ANALYTIC MODELING: THE STAR MODEL

The star is used as a baseline to which the performance of the MP/C is compared. It is chosen because in this case (the MP/C as a general-purpose machine) the star differs from the MP/C in exactly that characteristic which is under study, interprocess communications. (Note, however, that, unlike the MP/C, the star is limited to a relatively small number of processors, owing to the high degree of connectivity required of its central node.) In a star, the kernel is run on the central node, and each processor has a direct link to the kernel processor. Figure 7 shows the star model, drawn as a linear structure, to emphasize the structure common to the MP/C. As in the MP/C, each job runs on a processor for one unit of time (average). Unlike an MP/C job, it then goes directly to the center ($P_0$) and executes for $\alpha$ units of time, after which it returns to its processor.

H. ANALYTIC MODELING RESULTS

H.1. Processor Utilization

We first consider the utilization of user processors. We used a queuing network solver program, PNET, developed by Bruell and Balbo.\(^5\)\(^6\) It is a PASCAL program, running on a VAX UNIX, and (on our configuration) it can handle only up to 8 processor MP/C models. Figure 8 shows the productive utilization of the user processors, $P_0$ through $P_7$, in an 8-processor MP/C (one kernel, seven users), plotted versus the workload parameter $\alpha$. Recall that $\alpha$ models the ratio of preemption time to computation time. These curves justify the claim that the important range of workload is $\alpha < 1$. The curves show that, in the model, the farther a processor is from $P_0$, the lower its productive utilization is. Although this is a straightforward result for the model, we show in Section L that this is not exactly the case in a real MP/C.

We choose to observe productive utilization, rather than total utilization. Figure 9 shows an example of total utilization of three user processors. Some of the processors are heavily utilized even for high values of $\alpha$, but this high utilization only reflects the fact that they are preempted for long durations, not that they are productive. Total utilization in this case obscures the results and makes it noncomparable to the star, where there is only productive utilization. Hence, in the following "utilization" refers only to productive utilization.
The information from Figure 8 is condensed into a single curve of average utilization in Figure 10. It also contains the average utilization curve for the 8-processor star (one center, seven users). Since all processors in the star are equal, the average is also the curve of each of them. For reference, a third curve (marked [MP]) shows the utilization of an "ideal" multiprocessor. By ideal we mean that no interference between processors exists, and this can be viewed as modeling seven processors, each having its own I/O server. The utilization is defined there as \(1/(1 + \alpha)\), because 1 is the computation time and \(1 + \alpha\) is the total time.

Figure 11 shows the (productive) utilization of the MP/C normalized by that of the star. Figure 12 shows all the curves for \(N\), the number of user processors, \(2 \leq N \leq 7\). Note the following six phenomena:

1. The curves show a dip in midrange. In this area, the performance is very sensitive to \(\alpha\). Computation and I/O
times are about equal, and the extra delay of the MP/C is noticeable.

2. The dip deepens when \( N \) grows. Naturally, when there are more processors, more overhead is incurred.

3. The dip moves to the left as \( N \) grows. It takes smaller values of \( \alpha \) to achieve the same degradation when there are more processors to contribute to the overhead.

4. The left side has an asymptote from above \( \rho_{MP/C}/\rho_{star} = 1 \).
   When I/O time is marginal, the effect of preemption is reduced, thus the MP/C and the star are getting closer to being equivalent. (They are equivalent for \( \alpha = 0 \).)

5. The convergence to the left asymptote is very slow. This is because another effect makes the MP/C slightly worse than the star. In the star, each job has to wait in queue only in the central node. In the MP/C model, a job has to repeatedly wait in queues in each of the intermediate stations. The additional queuing causes this small loss of utilization, which exists throughout the \( \alpha \) range, but becomes the dominant factor only for small \( \alpha \).

6. For high values of \( \alpha \) we observe the same asymptote, \( \rho_{MP/C}/\rho_{star} = 1 \). The kernel is the cause of congestion, and all jobs spend most of their time in the queue of the kernel. Hence, the difference between the MP/C and the star becomes unimportant. As observed above, both the MP/C and the star perform rather poorly for \( \alpha > 1 \).
   Thus, this part of the curves may be ignored altogether.

To summarize, note that at no point is the utilization of the MP/C better than that of the star. On the other hand, for all the cases tested, it was always more than half that of the star. This suggests that the MP/C, although it is much more limited in communication paths to the kernel than the star, has at least very similar performance. And recall that some of the other MP/C advantages over the star are that all processors require at most two links to other processors (whereas the kernel in the star requires \( N \) links), and that the MP/C can be decomposed recursively into smaller MP/C subsystems. We have suggested elsewhere under what circumstances the performance of the MP/C might be much better, as well as what the other MP/C advantages are.\(^1\)

Figure 13 shows one additional measure. The minimum relative utilization at the dip is plotted versus \( N \), the number of other processors. The utilization of each processor is given by \( \rho_{MP/C}/\rho_{star} \).
of processors. This curve gives a lower bound on the performance of the MP/C, as compared to the star. Although it is speculative to conjecture how this curve might extrapolate, note that it shows a tendency to flatten out as $N$ grows. This gives rise to the conjecture that even for higher $N$ values the worst-case performance of the MP/C would not be much below that of the star. Also, note that star architectures with high $N$ are rather infeasible in general, owing to the high degree of connectivity required of its central node, whereas no similar requirement applies to the MP/C, since it is a simple linear structure.

H.2. Processor Utilization—Nonuniform Case

As stated at the end of Section F, an interesting question is the performance when not all users pose the same workload requirements. Indeed, this is probably also the more realistic case for a general-purpose, multiple-user environment. We describe here the results of the modeling when the workload consists of a geometric series, where each processor has its $\alpha$ parameter twice that of its neighbor. We consider the cases both of an increasing series and of a decreasing one.

As suggested above, intuitively we would expect that the best arrangement is the decreasing one, that is, when processes with high $\alpha$ are allocated to processors near the kernel, and those with low $\alpha$ to the processors further away.

As suggested above, intuitively we would expect that the best arrangement is the decreasing one, that is, when processes with high $\alpha$ are allocated to processors near the kernel, and those with low $\alpha$ to the processors further away.

Figure 14 shows the utilization of the MP/C, relative to the star’s, when the $\alpha$ values increase with $i$, the processor index. Figure 15 is the same for a decreasing series. The $\alpha$ values used in the graph are the average over all $\alpha_i$, $0 < i \leq N$ (this is not really important, since same $\alpha$ is used for both the MP/C and the star, and a different calculation of $\alpha$ would only shift the graph sideways). The results for low $\alpha$ values resemble the uniform case above. However, for high values of $\alpha$, we get results that are just the reverse of what might be expected. This phenomenon is commented on in the end of Section L, when it is compared with the simulation results (which do conform to intuition).

H.3. Cycle Times

As is explained in Section H, the first error introduced by the MP/C model causes overestimation of cycle times. Hence, it is reasonable to assume that the cycle times predicted by the model are only upper bounds on the real cycle times. The model prediction is depicted in Figure 16 by the curve marked “nonmodified cycle times.”

In an attempt to derive the complementary lower bounds, we consider the following observation. The cycle time of the user job from $P_i$ consists of the following:

1. Service time (mean = 1) and waiting time on $P_i$.
2. Service time (mean = $\alpha$) and waiting time on all $P_j$, $0 \leq j < i$, that is, on $i$ processors.

On the real MP/C, however, the cycle is composed of:

1’. Service time (mean = 1) and waiting time on $P_i$.
2’. Waiting time to seize the kernel.
3’. Service time on the kernel (mean = $\alpha$).

We assume that item 1 models item 1, and that the sum of all waiting times in item 2 models item 2. Then the balance is only one service time on the kernel in the real MP/C, but $i$ times the same service period in the model. This analysis suggests that the cycle times predicted by the model are too long.

Although this over-analysis is simplistic, we can use it to modify the cycle times by subtracting $i - 1$ kernel service pe-
iors from the total cycle time. This does not produce correct cycle times, but we assume that the modifying is an overshoot, and that it yields a lower bound.

Applying this procedure on the results described above, we obtain the modified cycle times curve in Figure 16. Thus, the model prediction provides an upper bound, and the modified cycle times are a lower bound. As we show in Section 1.2, simulations (whose purpose is to validate these results) show that the modified cycle times are indeed an optimistic prediction.

I. ANOTHER ANALYTIC MP/C MODEL

We describe a second model for the MP/C. It is based on the method of surrogate delays of Jacobson and Lazowska,16 as applied by Heidelberg and Trivedi14 and on Cobham’s analysis of priority queues.9,15

Each user processor, P, 1 < i < N, has one job associated with it, Ji. This job cycles between P, and the kernel. The kernel is a nonpreemptive priority queue, to model the fact that in the real MP/C the kernel serves the nearer processors first. Thus, Ji is assigned priority N - i. The cycle time consists of the following components:

1. Service time ri at Pi, with mean of 1 unit
2. Delay d, at P, due to preemption by Ji, i < k < N
3. Waiting time si at the kernel
4. Service time αi, at the kernel, with mean α (either uniform for all Ji, or variable)

Job Jk is preempted by Jk for αk, the average time Jk is served by the kernel, in each cycle of Jk, Ji stays at P, for 1 + di units of time. During that time, Jk makes λk (1 + di) cycles, where λk is its throughput. This delay is accumulated for all k > i. Hence,

\[ d_i = (1 + d_i) \sum_{k=i+1}^{N} \alpha_k \lambda_k, \quad 1 \leq i \leq N. \]  (1.1)

Solving for di,

\[ d_i = \sum_{k=i+1}^{N} \alpha_k \lambda_k / \left(1 - \sum_{k=i+1}^{N} \alpha_k \lambda_k \right), \quad 1 \leq i \leq N. \]  (1.2)

The waiting time in the kernel, si, is composed of two identifiable parts, the mean residual life R of the job in service upon arrival of Ji at the kernel, and the time Ji has to wait for all jobs of higher priority to be served. The residual life has been shown to be:

\[ R = \sum_{k=1}^{N} \lambda_k \bar{x}_k^2 / 2, \]  (1.3)

where \( \bar{x}_k^2 \) is the second moment of service time at the kernel for Jk. If we assume exponential distribution of service times with mean \( \alpha_k \), then the second moment is 2\( \alpha_k^2 \), and

\[ R = \sum_{k=1}^{N} \lambda_k \alpha_k^2. \]  (1.4)

During the waiting time \( s_i \), any job \( J_k, k < i \), with higher priority than \( J_i \) that arrives at the kernel is served before \( J_i \). The delay incurred is \( \alpha_k \) per cycle, and there are \( s_i \lambda_k \) cycles of \( J_k \) during \( s_i \). Hence,

\[ s_i = R + \sum_{k=1}^{i-1} \alpha_k \lambda_k s_i. \]  (1.5)

Solving for \( s_i \),

\[ s_i = R (1 - \sum_{k=1}^{i-1} \alpha_k \lambda_k). \]  (1.6)

We replace the queue at the kernel by a delay server (or infinite server) for each job with mean service (delay time for Ji of \( s_i + \alpha \)). We further replace the user processor Pi by a delay server with mean delay time of \( d_i + 1 \). Thus,

\[ \lambda_i = 1/(1 + d_i + s_i + \alpha). \]  (1.7)

The model is solved by iteration. Consider the vector \( x = (d_1, \ldots, d_N, s_1, \ldots, s_N) \). The initial condition is

\[ x = 0. \]  (1.8)

The vector of throughput results of iteration \( m \), \( \lambda^m = (\lambda_1, \ldots, \lambda_N) \), is used as input for the calculation of iteration \( m + 1 \). Since \( \lambda^m \) is a function of \( x^m \) (as per eq. (1.7)), and since \( x^{m+1} \) is a function of \( \lambda^m \) (as per eq. (1.2), (1.6)), we can say

\[ x^{m+1} = f(x^m). \]  (1.9)

Thus, we solve a fixed point problem \( x = f(x) \) in 2N dimensions. Note that equations (1.2), (1.6) are not guaranteed to produce stable, converging results always. In practice, we found that for \( \alpha \leq 1 \) equation (1.1) and (1.2) always converge. As will be shown in Section L, where the results are compared with simulation, this model produces fairly accurate results. However, for \( \alpha > 1 \), the model converges to the wrong results. It is possible that this is due to the fact that the model is based on continuous-time analysis, rather than discrete. But recall again that the interesting range is \( \alpha < 1/N \).

J. SIMULATIONS: SIMULATION METHODOLOGY

Since the analytic modeling we have described involves some approximations, simulations are necessary to validate the results. We show that some, though not all, of the analytic results are validated by the simulations.

J.1. Simulation Language

In order to minimize the programming effort required, a simulation language is used. Of the languages that were available to us at the time, SIMSCRIPT19 and GPSS,18 the latter was chosen because of relative simplicity and ease of learning. The disadvantage of using a language is that its flexibility is limited; that is, it may not be possible to build an exact model.
1.2. Simulation Models

In the MP/C simulation model, a single job is associated with each user processor. That job holds the user processor for a period of time (referred to as computation) that is drawn from an exponential distribution. Then it moves to the kernel processor, and competes with other jobs for its service. Another fact should be considered: On the MP/C, if two or more jobs request service, the kernel serves the nearest processor first. To account for that in the simulation model, each user job is assigned a priority level higher than that of all processors further away from the kernel (that is, to its right). In other words, if \( P_0 \) is the kernel, the priority of \( P_i \) is \( n + 1 - i \), \( 1 \leq i \leq n \).

When the kernel processor is seized by a user job from processor \( P_i \), all the processors between \( P_i \) and \( P_{i-1} \), inclusive, are halted for the duration of service. The service time on the kernel is distributed exponentially, and its mean is \( \alpha \) times the computation time on the user processors. \( \alpha \) can be the same for all user jobs (the uniform case), or it can vary. Once service on the kernel is finished, all the halted processors resume operation, and the job returns to its user processor.

The star model is similar, with two differences. First, no halting of intermediate processors is necessary. Second, all user processors are assigned the same priority level.

We have listed both models elsewhere.²

1.3. Method of Statistical Sampling

In order to generate a large sample for each experiment, we use the method of batches. In this method the simulated system is first allowed to achieve its steady state, and then a single long run is divided into time periods, or batches, and statistics are gathered for each batch. This method needs less computing than the method of independent replications, in which the system is restarted for each replication, and thus the extra time for the decay of the transient response has to be repeated many times. The third known method, regenerative simulations, is not easily applicable here, as no relatively frequent regeneration Markovian state can be identified for either the MP/C or the star. These methods are discussed by Kobayashi and by Sauer and Chandy.¹⁷,²¹

1.4. Length of Transient Response

The length of the transient response was determined empirically from a series of pilot runs. Figure 17 represents the behavior of two of the performance measures used, cycle time (for \( P_i \)) and utilization (of \( P_7 \)). These are accumulated statistics, such that every point in the graph represents the average of the observed measure from the beginning moment until the current time. The graph also shows the point chosen for the beginning of the first batch. It is clearly beyond most of the transient response.

1.5. Batch Length

The length of each batch was similarly determined by pilot runs. Figure 18 shows the behavior of the two measures during some long batches. In the beginning of each batch, all the statistics accumulated up to that point are purged, and accumulation is restarted. Hence, the sharp peaks at batch start represent random values, averaged over very short time periods, and the curves flatten out after some time. The batch length is chosen such that the transient effect of restarting the statistics accumulation has been damped. The batch end points are marked with vertical lines.

1.6. Number of Batches

The number of batches required is usually determined by the desirable confidence interval and by the need to make
statistically significant distinctions between different quantities.\textsuperscript{20} Also, to be able to claim that a sampling distribution is approximately normal (Gaussian), one usually needs at least 12 samples. Using a simulation run of 12 batches, we got results that were either almost identical, or separated well beyond any statistical doubt. Figure 19 shows the distribution density function of the sample averages of processor utilization of all the experiments made with either fixed $\alpha = 100$, or variable $\alpha$ with values around 100. Figure 20 is the same for cycle times. Note the narrow Gaussian "bells," the large distances between some, and the almost complete overlap of others. Note also that these are not the density functions of the distributions of the measures examined (they need not even be normal), but rather the distributions of their sample averages.

Table I—Comparison of pilot simulation and analytic results, star and MP/C with 7 user processors

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>Utilization, $MP/C$</th>
<th>Cycle Time, $Star$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pilot Simulations</td>
<td>Analytic Model</td>
</tr>
<tr>
<td>1000</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>100</td>
<td>0.9</td>
<td>1.0</td>
</tr>
<tr>
<td>1</td>
<td>0.9</td>
<td>0.79</td>
</tr>
<tr>
<td>0.4</td>
<td>0.89</td>
<td>0.65</td>
</tr>
<tr>
<td>0.01</td>
<td>0.97</td>
<td>0.92</td>
</tr>
<tr>
<td>0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>
K. PILOT SIMULATION RUNS

The first task of the simulation was to verify the general behavior of the analytic model, across different workloads ($\alpha$ values), for uniform $\alpha$ (i.e., same $\alpha$ for all user jobs). Pilot runs (i.e., sample size of one) were made at various $\alpha$ values. The results are summarized in Table I, and compared to the result of the first analytic model (Sections E through G). Although they do not constitute a statistical proof, the simulations are close enough to the modeling results to suggest that the analytic model provides reasonable results.

Table II—Comparison of simulation and analytic results, star and MP/C with 7 user processors

<table>
<thead>
<tr>
<th>Load</th>
<th>Cycle Time</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Star</td>
<td>MP/C</td>
</tr>
<tr>
<td>Simulations, $\alpha=0.1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>uniform</td>
<td>1.17</td>
<td>1.47</td>
</tr>
<tr>
<td>Analytic Models (absolute and relative to simulations), $\alpha=0.1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>uniform, modified</td>
<td>1.18 (101%)</td>
<td>1.65 (112%)</td>
</tr>
<tr>
<td>uniform, non-modified</td>
<td>*</td>
<td>1.95 (133%)</td>
</tr>
<tr>
<td>uniform, model 2</td>
<td>*</td>
<td>1.50 (91%)</td>
</tr>
<tr>
<td>Simulations, $\alpha=1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>uniform</td>
<td>6.95</td>
<td>9.60</td>
</tr>
<tr>
<td>Analytic Models (absolute and relative to simulations), $\alpha=1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>uniform, modified</td>
<td>7.00 (100%)</td>
<td>7.00 (73%)</td>
</tr>
<tr>
<td>uniform, non-modified</td>
<td>*</td>
<td>10.00 (104%)</td>
</tr>
<tr>
<td>uniform, model 2</td>
<td>*</td>
<td>10.95 (114%)</td>
</tr>
<tr>
<td>Simulations, $\alpha=100$ and variable $\alpha$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>uniform</td>
<td>698.1</td>
<td>940.4</td>
</tr>
<tr>
<td>increasing</td>
<td>502.7</td>
<td>998.6</td>
</tr>
<tr>
<td>decreasing A</td>
<td>502.7</td>
<td>371.6</td>
</tr>
<tr>
<td>decreasing B</td>
<td>1355.2</td>
<td>1001.2</td>
</tr>
<tr>
<td>Analytic Models (absolute and relative to simulations), $\alpha=100$ and variable $\alpha$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>uniform, modified</td>
<td>699.8 (100%)</td>
<td>581.9 (62%)</td>
</tr>
<tr>
<td>uniform, non-modified</td>
<td>*</td>
<td>882.0 (94%)</td>
</tr>
<tr>
<td>increasing</td>
<td>28.8</td>
<td>495.2</td>
</tr>
<tr>
<td>decreasing</td>
<td>1772.4</td>
<td>305.8</td>
</tr>
</tbody>
</table>

L. DETAILED SIMULATION

L.1. Results

The results of some detailed simulation runs are shown in Figures 19 and 20. As explained above, it is clear from those figures that ignoring confidence intervals and considering only the averages would not cause grave mistakes. Table II summarizes the simulation results for $\alpha = 0.1, 1, 100$. It also compares them to the results of the analytic model described in Sections E through G (both modified and nonmodified cycle
times) and (for $\alpha \leq 1$) the second model, Section H. Note that when comparing utilization, we consider the MP/C utilization divided (normalized) by that of the star; however, when considering cycle time, we use the MP/C cycle time in the denominator instead. This is because the inverse of the cycle time (rather than cycle time itself) is the measure of goodness here. Thus, for both measures, the higher the ratio, the better the MP/C as compared to the star. The entries marked “uniform” are the results with uniform workload on all user processors. The increasing case is when $\alpha_1 = 4$, $\alpha_2 = 8, \ldots$, $\alpha_5 = 256$ ($\alpha_1$ is the load on $P_1$, the processor closest to the kernel, etc.). This powers-of-two series of $\alpha$ values was chosen such that the load on the kernel (in terms of average time per job on the kernel) is similar to the “uniform,” $\alpha = 100$ case.

As expected, the MP/C in this case is worse (relative to the star) than in the uniform case. The decreasing case has two variants. In variant A, the loads are the reverse permutation of the previous case. That is, $\alpha_1 = 256$, $\alpha_2 = 128, \ldots$, $\alpha_5 = 4$. This time the MP/C performance is better than the uniform, $\alpha = 100$ case. However, the MP/C performance is so improved that the load on the kernel is significantly reduced. Hence, variant B was constructed with the following geometric series of $\alpha$ values: 6912, 3456, 1728, 864, 432, 216, 108, such that the average time per job on the kernel resembles that of the uniform, $\alpha = 100$ and the increasing cases. Nevertheless, the ratios are the same as in variant A.

The MP/C utilization can also be used to the lower bound provided by the first MP/C deterministic model of Section D:

$$\rho_{MP/C} = \frac{1/\alpha}{1/\alpha + N} = \frac{\beta}{\beta + N}$$

For $N = 7$ and $\alpha = 0.1, 1, 100$, $\rho = 0.59, 0.125, 0.00143$, respectively. The simulation showed slightly better performance (than this theoretical lower bound) for $\alpha = 0.1, 1$ but worse performance for $\alpha = 100$.

L.2. Discussion

One question naturally arises when these data are examined. Since the MP/C inherently demonstrates a priority mechanism, will a star with priorities behave like the MP/C? If not, what makes the difference? Introducing priorities to the star model takes effect only for high $\alpha$. There, it does not make the star imitate the MP/C. Rather, it causes livelock, or partial starvation. That is, the two or three jobs with the highest priorities get to use the kernel almost exclusively, whereas the lower priority jobs starve forever. The reason for this is best demonstrated in specific terms. Recall that for $\alpha = 100$ each cycle consists of, among other things, one unit of time (on the average) in the user processor and 100 units of service time in the kernel. Suppose the job of user 1 has just released the kernel, and the job of user 2 (having the next highest priority) seizes it. In the star, when the job from $P_2$ uses the kernel, $P_1$ is free to complete its computation part of the cycle and is ready to grab the kernel again once user 2 releases the kernel. Thus, most of the time the kernel is shuffled back and forth between $P_1$ and $P_2$, and the other users starve waiting for the kernel service. On the MP/C, when $P_2$ uses the kernel, $P_1$ is preempted. Thus, there is higher probability that a processor other than $P_1$ will be ready to grab the kernel once $P_2$ releases it. In short, $P_1$ can execute the (relatively small) computation part of its cycle only when no other processor uses the kernel. For seven processors and $\alpha = 100$, this is very rare indeed. On the other hand, $P_1$ of the MP/C, although it has the lowest priority, is never interrupted by anyone. This effect causes the inversion anomaly. In the simulations, the better-served processors are those further away from the kernel, the opposite of what may intuitively be expected. This anomaly of course can not be detected by the analytic model of Section F.

The inversion anomaly can be countered by a more elaborate preemption mechanism, for example, a minimal delay between successive preemptions, and by optimally arranged nonuniform workload.

The detailed simulation enables us to detect the inaccuracies of the analytic models. The two models show similar fit for $\alpha \leq 1$. Nonmodified cycle times of the first model are more accurate than modified cycle times for $\alpha = 1$ but less so as $\alpha$ decreases. As was explained in Section H, the second model does not work for $\alpha > 1$.

The reversal of the utilization figures at high $\alpha$ is an amplified version of the inversion anomaly. It seems to stem from the replacement of simultaneous preemption by successive service periods. Consider for example the increasing case. In the real MP/C, as well as in the simulations, when the user job from $P_1$ holds the kernel, no other processor is active, and recall that $\alpha_1 = 256$, the highest. In the analytic model, on the other hand, all processors, except $P_1$, are active at that time. Thus, the utilization in the increasing analytic model is higher than in the simulations. The error in the decreasing case is harder to figure out. The above explanation would have caused the same, rather than the opposite, effect.

Note, again, that these discrepancies are noticeable only at high $\alpha$. Consider Figures 11 and 16 and Table I. For $\alpha < 10$ it seems that both simulations and analytic modeling yield similar results, in both uniform and nonuniform cases. Since the analytic model breaks down only at high $\alpha$, and since, as it was emphasized above, this range of workload is not really important, we can ignore this problem.

M. CONCLUSION

We have discussed the MP/C performance under various circumstances. First, the performance of tree-structured algorithms was discussed. That application seems to be the most suited to the MP/C, and the MP/C performs probably better than any other architecture.

Next we examined the other side of the spectrum, the application of the MP/C to multiprogramming general purpose computations. This case serves as a worst case, or lower bound, analysis, for the MP/C. We have devised some analytic modeling techniques. Due to the complexity of the MP/C, only approximate analytic modeling was possible. Simulations were used to verify the results of the analytic modeling and to estimate their error.
We have found that the analytic models provide acceptable rough estimation of performance. Also, it was shown that, in the worst case, the performance of the MP/C is similar to that of the star multicomputer. That is, the relative advantages of the MP/C do not necessarily come at the expense of performance.

As we have suggested elsewhere, under the right conditions the MP/C has the potential to perform much better than other architectures. We are currently performing additional evaluation, comparing the MP/C to multicomputers using local networks and to multiprocessors using shared memory, by means of detailed emulations, simulations, and a complete implementation.

ACKNOWLEDGMENTS

We are grateful to Steve Bruell and Gianfranco Babo, who let us use PNET, and to Neil Polhemus, for helping with the simulations.

REFERENCES
