Measurement and Prediction of Contention in Multiprocessor Operating Systems with Scientific Application Workloads †

G. E. Bier
M. K. Vernon

Computer Sciences Technical Report #767

May 1988

† To appear in *Proc. 1988 Int'l. Conference. on Supercomputing,* St. Malo, France, July 4-8, 1988.

Measurement and Prediction of Contention in Multiprocessor Operating Systems with Scientific Application Workloads

George E. Bier
Mary K. Vernon

University of Wisconsin - Madison
1210 W. Dayton St.
Madison, WI 53706

Abstract

This paper examines the effect of contention for a single shared semaphore protecting critical regions in a multiprocessor operating system on system performance as the number of processors increases. A simple queueing model is used to make predictions as a function of the expected *demand* for the semaphore by each processor. Measurements are then presented for 2 and 4 processor configurations of the CRAY X-MP running an experimental version of UNICOS. The measurements indicate that demand is highly dependent on the workload, and that contention can be somewhat higher than predicted by the model due to process scheduling effects and request interdependencies. One interesting point which is revealed in the measurement results, is that the UNIX scheduler gives preferential treatment to jobs that make frequent requests for the semaphore.

1. Introduction

Recent increases in the processing power of supercomputers for numeric applications have been achieved partially by increasing the number of processors in the machine. Current systems scale to 2-4 processors, and designs for the near future scale to 8, 16 or perhaps even 64 processors. Unfortunately, there are factors that limit how much benefit is gained by each additional processor. In particular, contention between processors for shared resources can cause performance degradation. Contention can manifest itself in many ways. For example processors might compete for memory, for a shared bus, or for i/o devices. In this paper we examine contention for a software shared resource, the operating system. To our knowledge this issue has not previously been studied in any detail.

Operating system contention exists because of *critical regions*, regions of code that require exclusive access. On a uniprocessor system, only one user process can generate an operating system request at a time. Although the possibility for contention between a user process and system process exist (due to interrupts), there is no need to provide protection between user processes. On a multiprocessor, processes on different processors can make operating system requests concurrently. When this occurs,

This work was supported by CRAY Research, Inc., and by the National Science Foundation under grant number DCR8451405.

there is a possibility that requests will have to wait. If the waiting time cannot be filled by running a different process on the processor, then the waiting time decreases the computing power and thus the throughput of the system.

The redesign of an operating system is a formidable task that requires substantial performance pay-offs to justify the effort. It is known that scientific applications typically spend much less time using operating system services than general-purpose data processing and interactive workloads (perhaps 5-15% as compared with 40-50% in a typical UNIX environment [PaMa86, BaBu84, Fede84]). For these reasons, the trend in numeric supercomputers has been to port a uniprocessor operating system to the multiprocessor environment by identifying and protecting all the critical regions with a single semaphore. "System tuning" is then used to minimize time spent in critical regions. The disadvantage of this method is that the critical regions are not optimized for parallel execution.

This paper investigates the performance implications of having a single semaphore protecting the critical regions in supercomputer operating systems. We first develop a model of contention that assumes requests occur at random times. The model indicates that if each application spends 5% of its time executing critical sections of the operating system, then the degradation in processing power due to operating system contention is about 10% with 16 processors. We then present some preliminary measurement data that indicates actual demand is highly variable, and that process scheduling and interdependencies in the requests for the operating system semaphore can lead to somewhat larger decreases in processing power than the model predicts.

2. A Simple Model of Contention

We make the following assumptions in developing a simple analytical model of operating system contention. The first assumption is that requests occur randomly. That is, there is no interdependency in the times at which different processors request the semaphore. Second, the interrequest time for the semaphore from each processor has the same mean. This assumption implies that there is no statistical difference between the applications executing on each processor. Third, requests are assumed to use a single shared semaphore, and are handled first-come first-serve. Finally, the duration of the semaphore holding time is exponentially distributed with a specified mean value for all processors.

The only input to the model is the operating system *demand per processor*. The demand is the percentage of execution time a processor spends executing critical regions. If the mean interrequest time is denoted by Z and the mean duration is denoted by $\bar{x}$, then the demand is:

$$demand = \frac{\bar{x}}{\bar{x} + Z} * 100\% \qquad (1)$$

The queueing model in Figure 2.1 is used to model the system just described. The delay center at the top of the diagram models the time a processor executes between requests for the semaphore. The server at the bottom is the operating system semaphore. Each customer in the model represents a processor executing application processes. The results of the model depend only on the ratio of semaphore holding time to the interrequest time. This ratio can be computed from the input parameter, *demand*, defined above.

Performance measures computed from the model include the mean time a request spends waiting for the semaphore (W), the average utilization of the semaphore (U), the mean number of processors executing in user mode (n), and the throughput of the system (Λ). We are primarily interested in the average effective computing power of the system. Effective computing power can be calculated from the number of

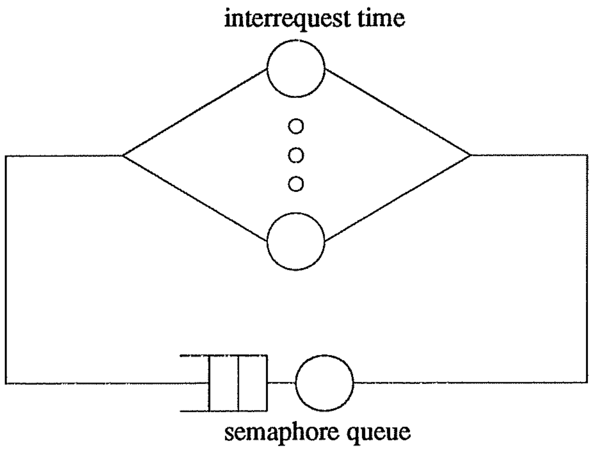


Figure 2.1: Queueing Model of Operating System Contention

CPUS (N) and the measures from the model using the formula:

$$\textit{Effective Computing Power} = N - \Lambda \times W \tag{2}$$

Alternatively, effective computing power can be calculated by $n + U$. However, we will find the above formula more useful for the measurements studies reported in section 3.

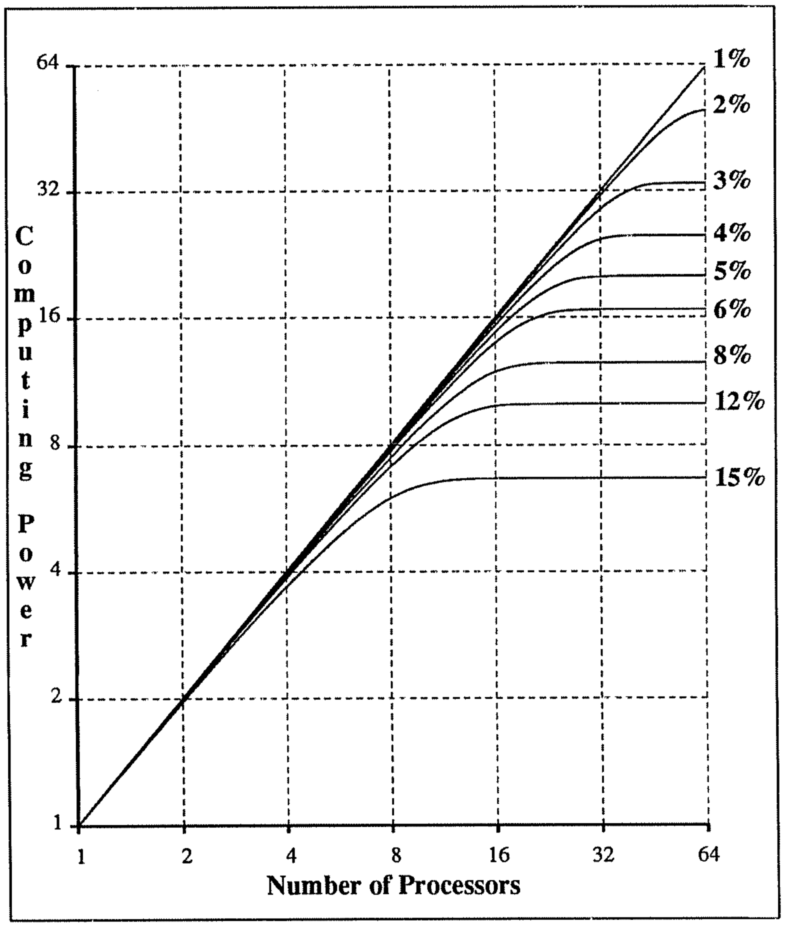
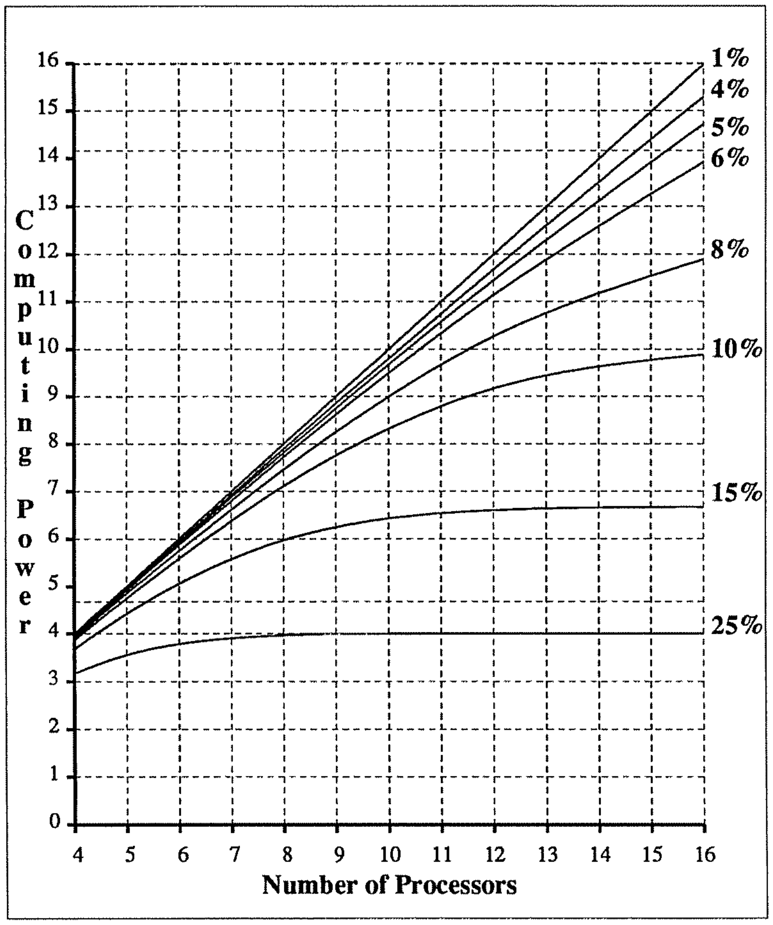


Figure 2.2: Predicted Effective Computing Power as a Function of the Number of Processors and % Demand

Figure 2.2 is a graph of the effective computing power as a function of the number of processors in the system, estimated by solving the model for demands ranging from 1 to 15 percent. The figure shows that a 1% demand does not reach saturation with up to 64 processors. However, for a 5% demand, operating system contention becomes significant at 16 processors. Note that once saturation is reached, adding more processors does not increase computing power because the operating system is limiting the system throughput.



Figures 2.3: Predicted Effective Computing Power as a Function of the Number of Processors and % Demand

Figure 2.3 is an expanded view for the range of 4 to 16 processors. This range is particularly interesting because it represents the sizes of current or soon to be available supercomputers intended for scientific applications. For a 4 cpu system, appreciable degradation in effective computing power occurs only for demands of 15% or higher. At 8 processors, demands above 8% lead to the loss of more than 0.5 processors in computing power. At 16 processors, the 4%, 5% and 6% are delivering effective computing powers of 15.3, 14.7 and 13.9, respectively. If the time spent in supervisor mode is 5% or more per processor then a 16 processor system will lose more than 1 high-performance cpu to contention.

3. Preliminary System Measurement Experiments

We have available time on a CRAY X-MP system for measurement studies of operating system contention. The operating system used in the study reported in this paper is an experimental version of CRAY UNICOS†. UNICOS is a version of UNIX† that has been ported to a multiprocessor environment.

Probes were added to the experimental version of UNICOS to measure the demand for the operating system semaphore. We wish to emphasize that the instrumented kernel used for the study is an experimental version. The description and results that follow may not agree with any released version of the operating system.

3.1. The Operating System

The experimental version of UNICOS used for our measurements has a single operating system semaphore used for protecting kernel critical regions. Multiple processors can be in supervisor state, but because critical regions are protected, only one processor can be in a critical region. Nine critical regions were identified that are protected by the operating system semaphore.

A test and set instruction is used to acquire the semaphore. If the test and set instruction is unsuccessful, a processor busy-waits. Requests waiting for the semaphore are served in random order.[1] A waiting processor cannot schedule a different process to run because the scheduler is protected by the semaphore. Even if scheduling a different process to run were possible, it may not make sense if context switching is more time consuming than the expected wait.

There are many possible paths through the operating system. A process might acquire and release the semaphore several times while remaining in kernel mode. This possibility introduces interrequest dependencies, a violation of one of the assumptions made for the simple model. One of the aims of our measurement study is to assess the impact of these dependencies.

† UNICOS is a trademark of CRAY Research. UNIX is a trademark of AT&T.

[1] Note that system throughput is the same for first-come first-serve service (assumed in our model) and random order of service.

3.2. Implemented Measures

In this initial study of the demand on the operating system semaphore, a minimal set of measures are obtained. These measures are: 1) the rate that requests are handled (the throughput or frequency of requests), 2) the mean time a request holds the semaphore (the duration), and 3) the mean time a request waits to acquire the semaphore (the waiting time). Higher moments of these measures were not obtained. We comment on this further in section 3.4. An important aspect of the duration is that it is not the length of time from when a request is made until it is satisfied, but the length of time the semaphore is held. For example, in the case of a read request, the duration of the request is the time it takes to place the request on the i/o processor queue. The semaphore is then released. The waiting time of a request is the time spent in a busy-wait loop because the semaphore is unavailable.

The experimental UNICOS kernel was modified to obtain three values from which the above measures can be computed, for each of the nine critical regions. One value is the total number of times each region was entered during a measurement interval.[2] The throughput for each region is then the total number of times the region is entered divided by the length of the measurement interval. A second value reported by the modified kernel is the total time spent in each critical region during the measurement interval. The average duration of a request for the region is then the total time in the critical region divided by the number of requests that occurred in the interval. The last value reported for each region is the total time spent busy-waiting before entering each region. The mean waiting time for each region is the total time spent busy-waiting before entering a region divided by the number of entries into the region.

To collect the data, code was added at the beginning and end of each critical region. Total waiting time is determined by recording the value of a hardware clock following the unsuccessful test and set instruction that begins a busy-wait loop. The clock is read again after the first successful test and set instruction that ends the busy-wait, and the difference is the wait time for the request. The wait time is then added to the sum of all previous wait times for the region. Incrementing a counter after each successful entry of a region gives the total number of times a region is entered. The duration of a request is the differ-

[2]We used a measurement interval of 10 minutes for the 2-CPU experiments and 5 minutes for the 4-CPU experiments.

ence in clock times from just after acquiring the semaphore to just before release. The duration for the current request is then added to the sum of the duration of all previous requests for the region. Note that the updating of the measures is protected by the semaphore, since each measure recorded is shared by all cpus. Note also that the implementation of this minimal set of measures is aimed at minimizing the overhead introduced by the probes.

The values for the nine regions are easily combined to give the total throughput, mean holding time, and mean waiting time for the semaphore. The total throughput is the sum of the throughputs for each of the regions. The overall mean holding time is calculated by summing the holding times for all nine regions and dividing by the total number of requests. The overall mean waiting time is given by summing the total waiting time for all nine regions and dividing by the total number of requests.

Given the experimental estimates of throughput (Λ), average duration ($\overline{x}$), and average wait time (W) of requests for the semaphore, we want to calculate the *demand* per processors (i.e. the percentage of time a processor would spend holding the semaphore if it was the only processor in the system). It would be incorrect to calculate the demand by multiplying the measured duration by the measured throughput and dividing by N (the number of processors) because the measured throughput includes waiting time. To calculate the correct percentage, we first apply Little's Law to obtain the average time for an execute-request-wait-hold cycle, and then subtract the mean waiting time and mean duration, to get the interrequest time per processor (Z):

$$Z = \frac{N}{\Lambda} - W - \overline{x} \tag{3}$$

The measured interrequest time is used to calculate the measured per-processor demand using equation (1) from Section 2. The measured demand is used as an input to the simple model to get the predicted effective computing power. The measured effective computing power is calculated from the measured throughput and measured waiting time using equation (2) given in section 2. The measured and predicted computing powers are compared to determine the accuracy of the model.

3.3. System Scaling

Our measurements give us the average demand *per processor*, not *per process*. We use the term *system scaling* to denote how demand on the operating system semaphore changes as the number of processors increases. The results presented in section 2 assume that the demand per processor remains constant as the number of processors is increased. We know that the demand of a particular *process* remains constant as the number of processors increases. Thus, provided there is sufficient work so that processors are never idle, a workload composed of identical processes should give roughly the same per-processor demand as system size increases. However, scheduling decisions and other factors may produce non-linear scaling effects for non-homogeneous workloads. We will examine both homogeneous and non-homogeneous workloads to validate the accuracy of the measures, and to examine the effect of system scaling on per-processor demand.

3.4. Measurement Results

We are interested in testing the validity of the results given in section 2 and in examining characteristic demands on the operating system for various types of workloads. The results of measurements given in this section are for a homogeneous compilation workload, a homogeneous executable workload with low i/o requirements, a non-homogeneous executable workload with higher i/o requirements, and a mixed workload of compilations and executables. We also examine the effects of doubling the workload, and the measurements of an idle system.

3.4.1. Compiler Workload

In this section we describe an experiment that measures the demand that FORTRAN compilations place on the system semaphore. We used a program that takes approximately 60 seconds to compile. This compilation is set up in an infinite loop so that when it finishes it will start again. To generate work for all the processors, 12 identical compilations were run simultaneously. The experiment was performed on a CRAY X-MP in both a 2-processor and 4-processor configuration. The results of the experiment are given in the first two rows of Table 1. The frequency (i.e. throughput), duration and waiting times are the total values for the nine critical regions.

From the table, we see that for this homogeneous workload, the mean duration remains relatively constant for the two and four processor configurations. It is also true that the mean duration measured for each of the nine critical regions that contain code whose execution is independent of the number of cpus, remained constant for the 2-cpu and 4-cpu cases. (This data is not shown in the table.) This increases our confidence in the correctness of the instrumentation.

The 10% drop in per-processor demand between the 2-cpu and 4-cpu results, is observed in all of the homogeneous workloads we have studied, and appears to be an important phenomenon. We are currently investigating the cause of this decrease. We surmise that there are sections of the operating system which are not critical regions, but which have more work to do when the number of processors increases.

Table 1: Measurements and Predictions for Workloads

# CPUS	Frequency (requests/sec)	Mean Duration (microsec)	Mean Wait (microsec)	Measured demand(%)	Effective Processing power measured	predicted	%difference
12 Identical Compilations							
2	3896	121	58	26.64	1.78	1.87	5.06
4	5940	125	163	24.40	3.03	3.21	5.94
20 Identical Executables							
2	42	99	24	0.21	2.0	2.0	0
4	72	87	14	0.16	4.0	4.0	0
15 Different Executables							
2	254	86	8	1.10	2.0	2.0	0
4	1244	119	58	3.76	3.93	3.98	1.2
30 Different Executables and Compilations							
2	694	157	70	5.59	1.95	1.99	2.05
4	1700	149	108	6.65	3.81	3.94	3.41
30 Identical Compilations (Effects of Swapping)							
2	3574	190	129	44.12	1.54	1.67	8.44
4	4462	202	374	38.69	2.33	2.44	4.72
Idle System							
2	151	41	11	0.31	2.0	2.0	0
4	127	38	9	0.12	4.0	4.0	0

Examining the waiting time for the two and four processor runs, we see that the waiting time has almost tripled. Note that the increase in waiting time explains why the frequency of requests has not doubled when moving from two to four processors.

The compilation workload yields a measured demand of roughly 25%. This is a high demand that has severe performance implications for systems with four or more processors. The last three columns in Table 1 compare the measured effective computing power of the experiment with the simple model's estimates for the measured demand. We see that the measured effective CPU power for 4 processors is 3.03. For this high-demand workload, a full CPU is lost to contention.

The model estimates are somewhat optimistic when compared with the actual system, predicting about 5% higher computing power. There are several possibilities for the discrepancy between the model and the actual system measurement. Experimentation with model estimates for large variance in the semaphore holding times indicates that this is an unlikely source of the discrepancy. We are currently investigating other possible sources of increased contention, including some known interdependencies in the requests for the semaphore. The key point is that the model slightly overestimates system performance.

3.4.2. Executable Workload I

The set of measures in this section are for a workload consisting of 20 processes. Each process is the exact same executable program. When a process finishes, it is immediately restarted. The program takes approximately 40 CPU seconds to run and has low i/o requirements. The third and fourth rows of Table 1 give the results of this experiment.

For this workload, we see that the demand is only about 1/5 of a percent. The results indicate that when moving from 2 to 4 processors, the average duration decreases by 10%, yielding additional decreases in the per-processor demand, and a decrease in the mean waiting time. We attribute this result to uncertainty in the measurements, since the sample sizes are relatively small for this low-demand workload. The measured and predicted processing powers are in agreement and show ideal performance. If executable workloads typically have such low demands, then 64-processor systems will not suffer degradation due to operating system contention for these workloads.

3.4.3. Executable Workload II

In the experiment in this section, the processes running are not identical. The workload is a job mix of 15 different executables. The execution times of the jobs range from 1.1 to over 300 CPU seconds. Two jobs have large i/o requirements, three have low i/o requirements, and the rest have no i/o requirements. All the processes are set up as infinite loops.

Rows five and six of Table 1 show that for this mixed workload, the demand per processor *increases* when moving from 2 to 4 processors. In fact, somewhat surprisingly, we observe nearly a five-fold increase in the frequency of semaphore requests. Further examination of this result reveals that this is a property of the UNIX scheduler. The scheduler penalizes jobs that have acquired CPU time and rewards jobs that have received little CPU time. Furthermore, the acquired CPU time measure decays with elapsed time. Jobs that make frequent system calls accumulate CPU time more slowly than jobs that make fewer calls. Also, short jobs are preferred by the scheduler, and each of these jobs makes system calls at start-up and termination. Thus, *frequent callers receive preferential treatment from the scheduler, increasing the per-processor demand on the semaphore when there are more processors.* This means that with a standard UNIX scheduler and any heterogeneous workload, increasing the number of processors can lead to higher contention than is predicted by the simple scaling assumed in the model of section 2.

The mixed workloads (in this and the next section) are the only tests where the average duration of requests changes significantly in the 2-cpu and 4-cpu tests. It appears that in this first mixed workload, the processes that are receiving preferential treatment make requests that hold the semaphore longer than the less frequent requesters. This causes the mean duration to increase.

The performance figures for this workload indicate that the per-processor demand for the two processor configuration is low, about 1%, and the effective computing power is nearly ideal. The four processor configuration has a substantially higher demand, showing the combined effects of the increased frequency and average duration of requests. Again, the difference between the measured and predicted effective processing power is small.

3.4.4. Mixed Workload

The second mixed workload is an experiment with the same 15 jobs as in the mixed workload of the previous section, with the addition of 15 processes each running a compile of a copy of the program used in the 15 executables mix. This produces a total of 30 processes executing concurrently, all set in infinite loops. Rows seven and eight of Table 1 give the measurement results.

This data also shows some increase in demand when scaling from 2 to 4 processors, due to preferential scheduling of frequent requesters. The effects of the scheduler are less dramatic than in the previous workload, since the current workload is more homogeneous (i.e., the compiler, which comprises half the processes, makes about the same demands on the system regardless of what it is compiling).

The measured effective processing power indicates that the system with 4 processors is losing 4.5% computing power to contention. The simple model estimates are 2-3% higher than the system measurements for this workload. Recall that the model estimates show more than 10% loss in effective computing power with 16 processors, for the measured demand of 5 to 6%.

3.4.5. Scaling the Workload

Increasing the number of processes within a workload increases the number of jobs the scheduler must handle and the amount of swapping due to context switching. To investigate the impact of these issues on our measurements, we ran the compiler experiment described earlier but increased the number of processes from 12 to 30. Rows nine and ten of Table 1 give the results of the experiment.

All the processes in the workload have identical characteristics. Thus, the per-process demand time should be identical for both the 30-compilation and 12-compilation workloads. In fact, when comparing the data in rows 9 and 10 with the data in rows 1 and 2, we see that the per-processor demand has increased from 26.64% to 44.12% for two processors and from 24.40% to 38.60% for the four processor configuration. We attribute the extra demand to two factors: 1) extra time spent in the scheduler and 2) swapping effects. The swapping effects appear to dominate.

We examined the average swap queue size for the two experiments. The queue sizes were 1 and

14 for the 12- and 30- compilation workloads, respectively. We also repeated the experiment with 8 and 15 compilations. The 8-compilation demand was about 25% for the two and four processor configurations. This is in very close agreement with the 12-compilation results. The 15-compilation workloads had demands of about 30%, which lie between the 12-compilation and 30-compilation experiments. From these experiments, we conclude that 1) swapping should be avoided to minimize contention for the semaphore and 2) swapping effects are minimal in the 12 compilation workload reported earlier.

3.4.6. Idle System

The last two rows of Table 1 reports the measurements of an idle system. The normal background daemons are running as they were for all previous experiments reported. Clock interrupts and scheduler invocations occur in sequence 60 times per second, and together account for nearly all of the activity measured in the idle system. Note that these interrupts occur a maximum of 60 times per second, but in the presence of active processes they occur much less frequently, as is clearly demonstrated by rows 3-4 of the table. The idle system and executable workloads with low i/o requirements yield very low contention for the semaphore, and nearly ideal effective computing power.

3.5. Preliminary System Measurement Conclusions

From the results of the initial measurement studies reported above, we reach several preliminary conclusions. First, the actual demand on the operating system semaphore is highly variable, and depends on the type of workload measured. Measured executable workloads have per-processor demands in the range of 0.1 - 4%. Compilation workloads have demands of about 25%. Combined compilation and execution workloads have demands of 5.6% and 6.7%. Second, the scaling of per-process demand as the number of processors increases for a fixed workload is complex. If the workload is homogeneous, per-processor demand *decreases* by about 10% when the number of processors is changed from two to four. If the workload is non-homogeneous and the operating system uses the UNIX scheduling algorithm, then per-processor demand for the operating system semaphore *increases* as the number of processors increases. This is because the UNIX scheduling algorithm gives preferential treatment to jobs that make more frequent requests for the semaphore. Third, the need to swap pages between memory and i/o devices increases the demand on the operating system semaphore, and should be considered in projecting system performance for large systems. Finally, the simple model in section 2 is optimistic by approximately 2 - 5% when compared with system measurements of realistic workloads. For workloads with very high demands the model's prediction is still within 10% of the measured value. The error in the predictions is at least partially due to known interdependencies in the semaphore requests that are not incorporated into the model.

4. Conclusions and Future Work

We have presented a simple model for evaluating the performance implications of using a single semaphore to protect the critical regions of a supercomputer multiprocessor operating system. This model predicts that if applications require the semaphore for less than 1% of their total execution time, the system can scale to 64 processors without appreciable degradation in peak computing power. If, on the other hand, application demand for the semaphore is 5%, the degradation in processing power due to contention for the semaphore is 10% when the system configuration includes 16 processors.

Preliminary measurement studies indicate that 1) the demands of actual workloads are highly variable, and 2) the actual degradation in computing power is somewhat worse than predicted by the simple model for a given demand. Some executable workloads have demands in the range of 1 - 2%. On the other hand, compilation workloads can present a demand as high as 25%. Interdependencies in the semaphore requests, which are not represented in the model, are at least partially responsible for the lower observed processing power than estimated by the model. These preliminary results lead us to conclude that operating system contention is a factor to be considered when designing supercomputers of four or more processors.

Three extensions to the measurement studies are needed in order to complete a more sophisticated model that can be used to project the effect of semaphore contention in larger configurations with confidence. The first extension is the implementation of measures that will allow us to uncover all of the reasons that actual contention is higher than contention predicted by the simple model. (We have determined that higher variance in the semaphore holding times and/or known interdependencies cannot account for all of the discrepancy.) The second extension is the implementation of measures, and/or experiments with additional workloads, that will allow us to more fully understand the precise way in which demand for the semaphore scales as the number of processors increases for a fixed workload. Finally, we need to measure a wider variety of workloads on a wider variety of supercomputers. Measurements at a typical supercomputer user site are highly desirable. We are currently pursuing these extensions.

Acknowledgements

The authors gratefully acknowledge Chris Hsiung at CRAY Research, Inc., for suggesting the topic of this paper, and Jim Harrell and Bruce Steger at CRAY Research, Inc., for their assistance in carrying out the measurements.

References

[BaBu84] Bach, M. J., Buroff S. J., "Multiprocessor UNIX Systems," *AT&T Laboratories Technical Journal*, Oct. 1984, Vol 63, No. 8. Part 2, pp. 1733-1750.

[Fede84] Feder, J., "The Evolution of UNIX System Performance" *AT&T Laboratories Technical Journal*, Oct. 1984, Vol 63, No. 8. Part 2, pp. 1791-1841.

[PaMa86] Paul, G., Martin, J. L. "Aspects of Performance Evaluation in Supercomputers and Scientific Applications",private communication