Proceedings of the 1991 Winter Simulation Conference
Barry L. Nelson, W. David Kelton, Gordon M. Clark (eds.)

ENVIRONMENT PARTITIONED DISTRIBUTED SIMULATION OF QUEUEING SYSTEMS

John Craig Comfort
School of Computer Science
Florida International University
Miami FL, 33199

ABSTRACT

Discrete Event Simulation (DES) is an extremely useful tool, with applications in manufacturing, computer science, management, engineering, and the military. Simulation programs tend to consume computer time in huge chunks. It is thus reasonable to attempt to execute these programs on super computers, or, better, on distributed multi-processor systems. The published results of these attempts have been disappointing, especially for an important subclass of DES models involving queueing systems. It is not even known how much performance improvement (speedup) is possible in a simulation of a general queueing system. In this paper, an extension of object oriented design is used to design computer systems to simulate specific queueing networks, and the performance of these systems is estimated through simulation. Substantial speedups were obtained for all three queueing networks. In addition, sensitivity tests were performed on communication and interface attributes of the system.

1 INTRODUCTION

A discrete system is one in which the state of the system changes only at discrete, separated instants of time. Discrete Event Simulation (DES) is the simulation of such systems. Substantial application of DES have been made in the fields of manufacturing, design (especially of VLSI components), computer system selection and analysis, and military logistics. In all these application areas, simulation programs have been implemented which have consumed incredible amounts of computing resources. As the systems simulated contain significant concurrency, it would seem reasonable either to run the simulation programs on supercomputers employing fast concurrent processors, or, with likely greater cost effectiveness, to run the simulations on distributed computer systems employing large numbers of processors. This last approach is called Distributed Simulation (DS). With a very small number of exceptions, the attempts to realize distributed simulations have been unsatisfactory.

All the application areas mentioned above have a common factor in that they contain (priority) queues of entities within the model. These queues may represent collections of parts waiting to be machined or shipped, logical signals waiting to be combined, or perhaps requests for service within a computer network. In addition, each simulation possesses an explicit (or in some cases, implicit) priority queue, the Event Set, of those events which have been scheduled but have not yet occurred. It is thus natural that a study of Distributed Simulation applied to queueing systems would provide insight into the nature of the overall problem of DES, and suggest ways in which the concurrency inherent in the real systems could be exploited in their simulations. Unfortunately, none of the empirical or theoretical results published has been encouraging, and the large question of the degree of parallelism inherent in Discrete Event Simulation remains open.

A pair of metrics will be used to measure the effectiveness of the methodologies employed. The first is the Speedup Factor, which, for a given simulation, consists of the ratio of the run time of the enhanced system to that of the original uni-processor simulation program. The second -- Processor Efficiency, sometimes denoted by alpha -- is the quotient of the Speedup Factor by the number of processors in the enhanced system.

Two basic strategies have been applied to the problem of mapping a simulation model onto a network of processors. In the classical strategy, the model is partitioned into components or subsystems, and the components are assigned to physical processors. In the alternative strategy employed in this research, the environment in which the model is run is partitioned into components, and the components are assigned to processors. Tasks (priority queue processing, random number generation, etc.) required to perform the simulation are identified, objects are constructed to perform these tasks, and selected instances of those objects are assigned to independent processors. This second strategy attempts to exploit parallelism latent in the system running the simulation, rather than in the model itself.

Empirical results suggest that the first approach can be quite successful in applications permitting a partitioning with a rather high level of computational granularity (the ratio of time spent in processors' local computation to time spent in message passing between processors.) Significant successes have been recorded (Speedup Factors of 28 in a 40 processor network by Lomow et al (1988)), but only in systems with granularities of four or higher. Simulations of pure queueing systems do very little computation per queue, so have inherently low granularity. In one study of a benchmark queueing system (here referred to as CR5), Wagner et al (1988) have presented a proof that no conservative simulation can achieve a speedup of more than 3.67, regardless of the number of processors employed.

Thus, in such systems of low computational granularity, environment partitioning seems appropriate. Jones et al (1989) has proposed what he calls Concurrent Simulation, in which the simulation processes' state space is updated in parallel with the scheduling of pending events. He has achieved some significant success (speedups of 2.3 with 4 processors on the CR5 benchmark), but it is not clear how to expand his technique to deal with large models. The approach here presented is an extension of that described by Rajagopal and Comfort (1989).

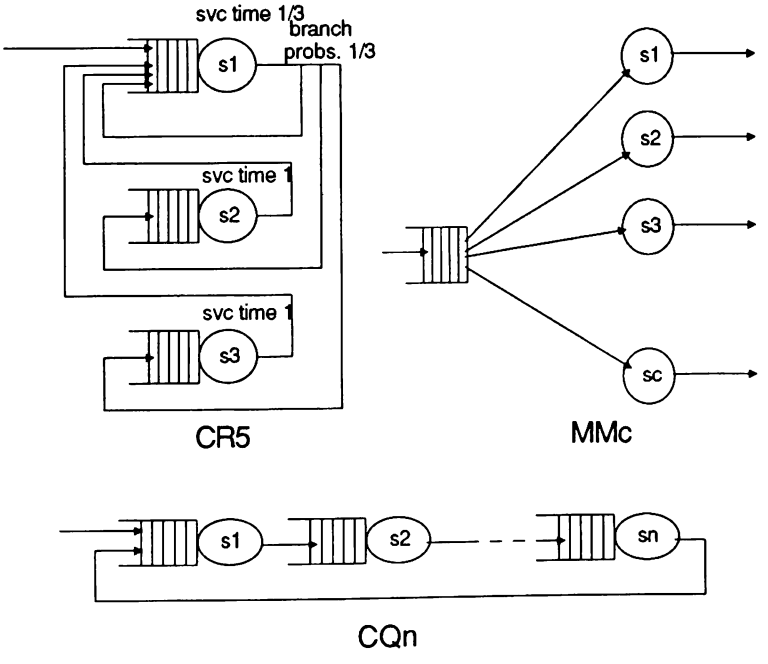

Figure 1. Three Kinds of Queueing Systems

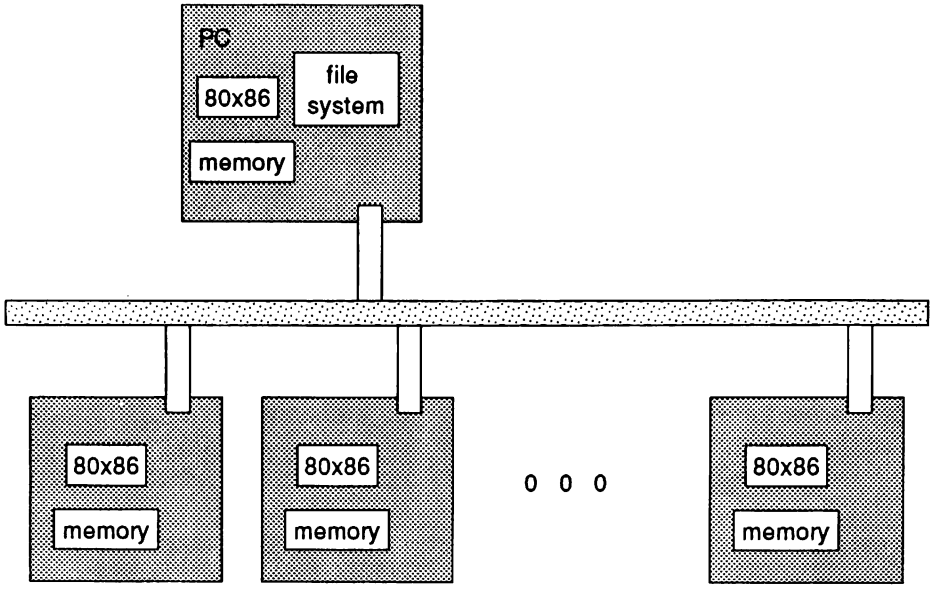

Figure 2. The Structure of the Simulation Computer

2 BASIS FOR THE RESEARCH.

2.1 The Queueing Models

Three queueing models were chosen to be investigated. The first is a slightly modified version of the CR5 benchmark mentioned above, a highly simplified model of a computer system containing a CPU and two I/O devices. The second is a closed circular queueing network containing n queues (CQn), which was included because it should permit easy model scalability -- that is, as n increases, it should be possible to maintain a relatively constant efficiency by employing k*n processors. The third is single level multi-server queue (called MMc), which was included because it is typical of the queues that appear in real simulations, and allows the investigation of a system under different object loadings, as the relative object loadings may be varied by (indirectly) varying the size of the event Figure 1.

2.2 The Target Hardware

The computer system visualized for running the distributed simulation consists of n plus 1 processor homogeneous computers, each with sufficient memory to contain the objects assigned to it. One of these processors, the master, is assumed to contain the controlling simulation program. In addition, it contains the file system and user interface for the aggregate system. The other processors, called ancillae, contain instances of object(s) to be employed in the simulation, and a software interface to process messages from the master. The processors are joined on a bus (different from the master's internal bus), whose functions include loading programs from the master's file space to the ancillae, and providing communication between the master and the ancillae. As only the master may initiate communication on the bus, there will be no contention to degrade bus transfer efficiency.

Specific timing data were obtained from a PC clone containing an 8086 processor running at 9.54 MHz., and 640K of memory. All processors in the system are assumed to have these attributes. The master/ancillae bus is assumed to contain 16 data lines, 8 address lines, and 3 synchronization lines. It is assumed to have a bus speed of at least 10 MHz.

2.3 The Target Software

All the programs employed were implemented in the object oriented extension of Borland International's Turbo Pascal (Version 5.5). It is assumed that the code for the individual objects,

together with interface software written either in Pascal or in assembly language, would be transferred to the ancillae, and that, interface details aside, the execution of the object instances on the ancillae would be identical to that on the master processor. The Pascal programs were written to be maintainable, rather than fast. No hand coding of any of the objects was performed.

2.4 Processor Coupling

To avoid unnecessary processor waiting, a cache has been associated with each remote object, and a cache management process (CMP) created to run as a foreground task while the object task remains in the background. A CMP is responsible for apprehending all messages from the master -- if the message serves as an input to the object, it is queued. If it requests an output, the output is taken from an output cache maintained by the object. (For an example of such a cache used to maintain the event set, see Comfort (1989)). The three different primitive object types -- statistics accumulator (STAT), priority queue (PRQ), and Random Number Stream (RNS) -- require slightly different cache organizations. STAT accepts requests for state statistic updates. If the queue becomes full, or if it is the end of a block, and statistics must be output, the master must wait until STAT has emptied its queue. If PRQ receives a request to schedule an entity, the request may be queued, unless the request requires that the cache must be updated. RNS is the simplest of the three objects -- it simply causes random numbers to be generated until its output cache is full.

2.5 Object/Instance Partitioning

The objects employed in the system can be assigned to the ancillae in various ways. The most obvious include assigning all objects to the master, and assigning each object to one ancilla. If either of these assignments produce acceptable results, then no more need be done. However, as the maximum speedups obtainable are 1 and 4 (and this assumes that the master and all object executions are completely overlapped) respectively, a finer partitioning will probably be desirable.

There are many random number streams employed in each of the target simulations -- thus assigning each stream to a different ancilla seems reasonable. Similarly, each simulation employs several priority queues, as well as an event set. Assigning each of these to an ancilla is also possible. Finally, the accumulation of statistics for one state of the simulated system will be independent of the accumulation of statistics for any other state, provided that there are separate invocations of the object for an entity entering and leaving the state, and that the time of entry of an entity into a state is maintained within the entity itself, and passed as a parameter.

Thus, even for the innocuous CR5 system, employing 4 random streams, 4 priority queues, and 12 states (6 client states -- waiting and served for each of the 3 servers, and 6 server states -- idle

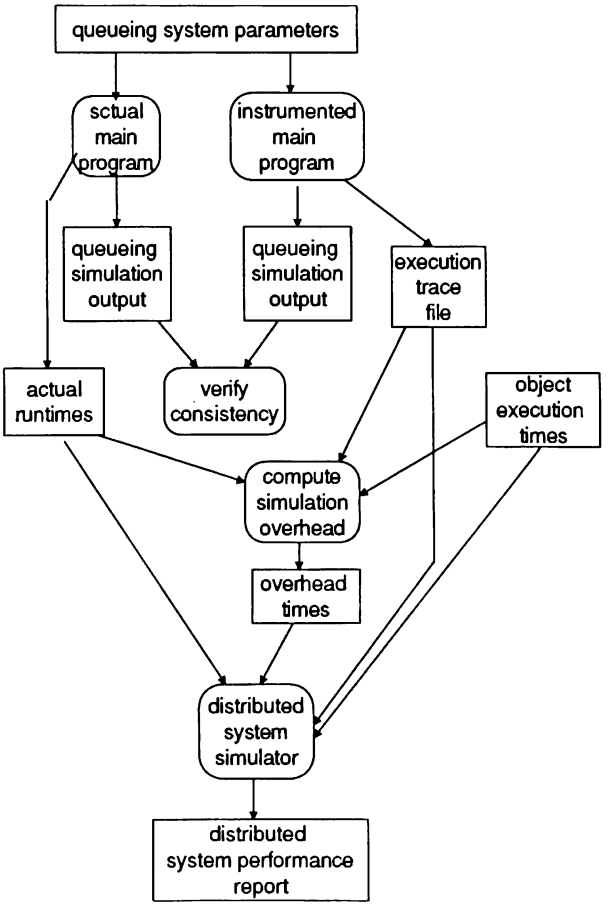

Figure 3. Data Flow Diagram of the Procedure

and busy for each of the 3 servers), it is possible to assign 21 (!) (master plus 20 ancillae) processors to its simulation. Whether it is desirable to do this, of course, depends on the magnitude of the speedup obtained.

3 METHODS

3.1 The Approach

The performance speedup obtainable was estimated by simulating the performance of a distributed computer system performing the simulation of the target queueing models. Simulation programs for the target queueing models were implemented, and instrumented to produce trace files containing parametric information about each object invocation. The information produced by the target simulation programs is verified, then used as input by another program simulating the distributed systems under evaluation.

Estimates for object execution times were determined by constructing independent programs to invoke the desired functions a substantial number of times (usually 100,000). After the effects of the controlling software were eliminated, confidence intervals for the functions' execution times were computed.

In addition to the object execution times, it was necessary to estimate the invocation time (time required for call/return) each instance, and the interface time (time required to transfer the required information to/from the ancillary cache) for each instance. These estimates were obtained either through other programs, performing their respective operations repeatedly, or from calculations based upon the bus timing data presented above.

However it was obtained, the mean value for each execution time for each object invocation, together with its communication and interface times, was inserted into a timing file to be used by the main simulation program. Subsequent sensitivity analyses were performed by varying some of these times.

The experimental procedure and its software and informational components are shown schematically in the data flow diagram of Figure 3.

3.2 The Experiments

For each queueing model, several simulation runs were made. The first run made use of the configuration containing only the master processor. In this simplest case, the observed Speedup Factor should theoretically be one. However, discretization errors (an integer clock is employed) result in a slight deviation from the ideal, with observed speedups ranging from 0.995 to 1.004. The second run made use a three ancillary processor configuration -- one each for the statistic object (all instances), the priority queue object, and the random generator object.

Subsequent experiments were guided by results obtained from previous runs. If an ancilla assigned to a collection of instances were heavily loaded (busy more than 90 per cent of the time) its instances were partitioned and reassigned. If two or more ancillae (implementing instances of the same object) were lightly loaded, their instances were coalesced and assigned to one ancilla.

For the CR5 system, additional runs were made, varying the communication and interface times, to investigate the sensitivity of the system's performance to these attributes.

4 RESULTS

In the following, a selection of the results obtained in this study will be presented. The results of experiments were made using different random number streams for the simulations were replicative of those shown, and have not been reported. The experiments concerning the system's sensitivity to variations in communication and interface time were performed only for the CR5 queueing system, as these results depend only on the target system architecture, and not on the system being simulated.

In all the experiments, the original simulation consisted of four or five blocks, each with a run time of between 8 and 15 seconds. To eliminate errors due to start up, results from the first block were discarded, and the mean values from the remaining blocks displayed.

4.1 Results from the CR5 System

Figure 4 shows the experimental process used to determine an effective assignment of instances to processors, and Figure 5 shows the specific performance obtained for all four configurations. When one ancilla was assigned to each object, a Speedup Factor of 1.57 was obtained. The heavy processor loading of the "stats" processor, and the somewhat less stringent loading of the "RNG" processor suggest that the functions assigned to these processors might be partitioned. One such partitioning employing ten processors produced a speedup of 3.92. To determine the relative loadings imposed by all the random number generation and statistical accumulation, each instance was assigned to an ancilla, resulting in a speedup of 7.20, but a processor efficiency of only .422. Pooling lightly stressed instances lead an eleven processor system, with a speedup of 7.33, and an efficiency of .666. While no claim is made that this performance is optimal, it certainly is respectable, especially considering previously reported results.

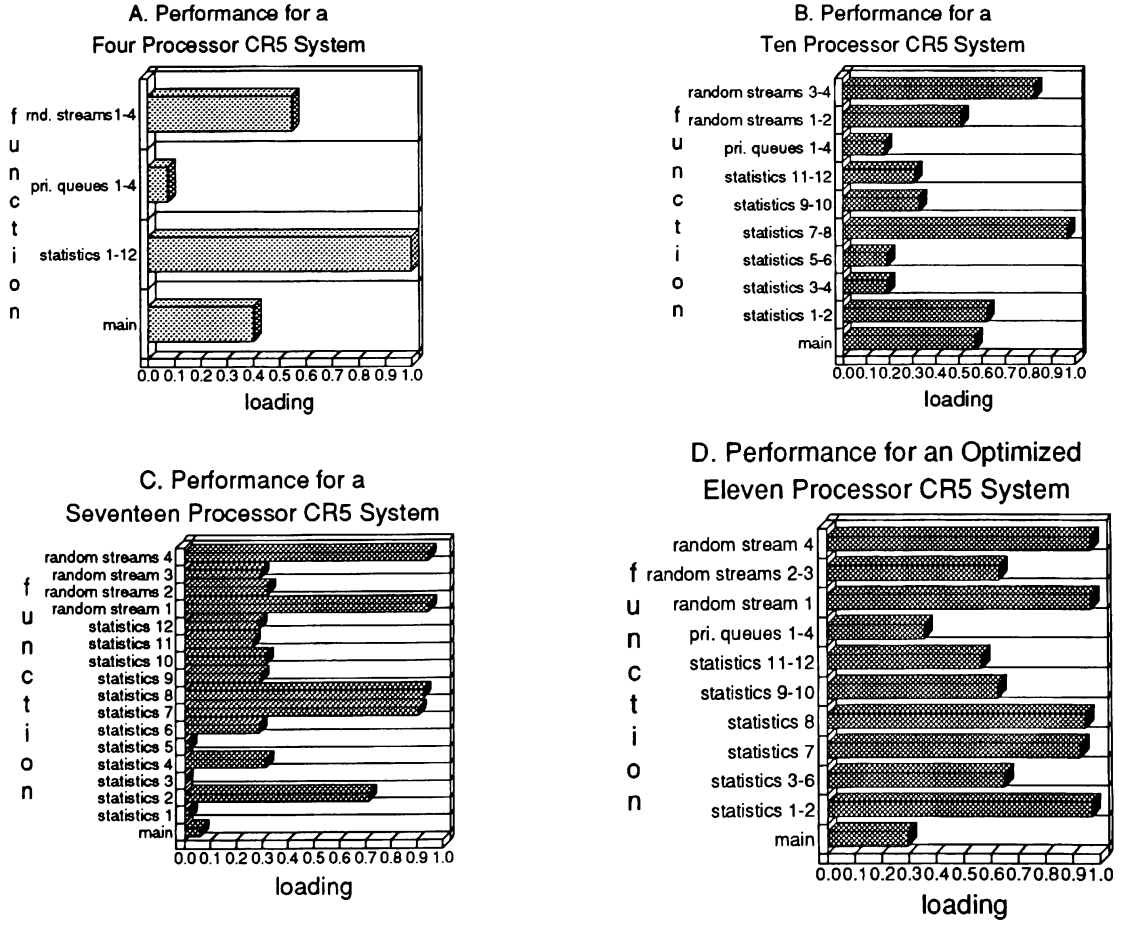

Figure 4. Processor Loading for Various CR5 Simulation Computers

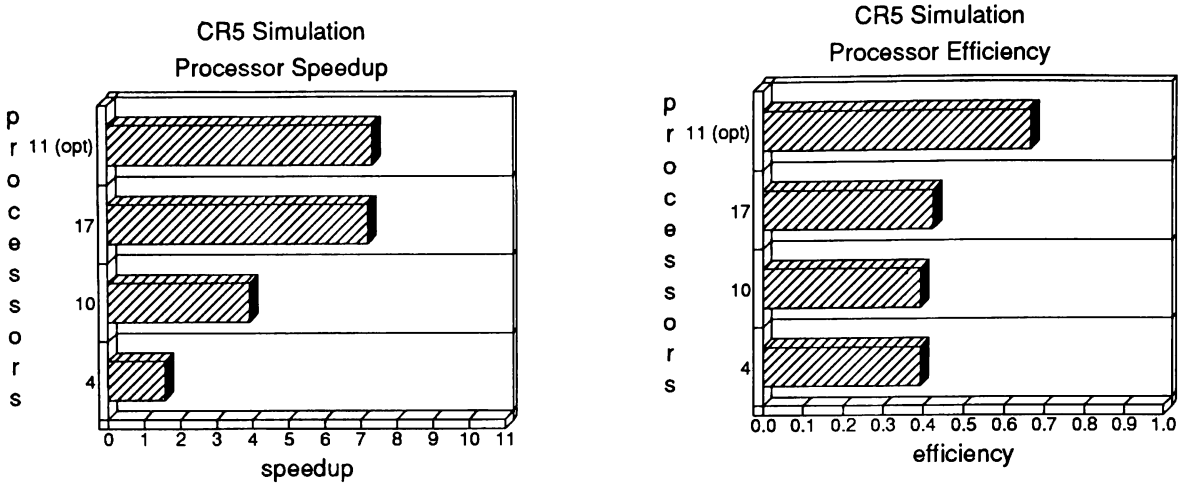

Figure 5. Speedup Factors for Various CR5 Simulation Computers

4.1.1 Communication/Interface Sensitivity

As the eleven processor system seemed to yield satisfactory performance, it was used as a target for an analysis of its sensitivity to variations in the communication and link speed. These (simulated) speeds were originally varied by the ratios of .5, 2, and 4 of the speeds used in the previous simulation. To obtain more information, in particular for the sensitivity to communication speeds, additional ratios of 3, then of 2.5 were employed. The results, presented in Figure 6, show that a variation of up to a factor of two in either parameter would not significantly excess of twice the values used would. An interesting result may be observed when the ratio is one half, for the resulting run time is slightly lower than that with the communication speeds actually employed. This phenomenon (not statistically significant) occurs because the master processor's time spent in "busy" state increased from 8 to 44 per cent, and master processor overhead, which had been hidden from the total run time through overlap, now becomes visible.

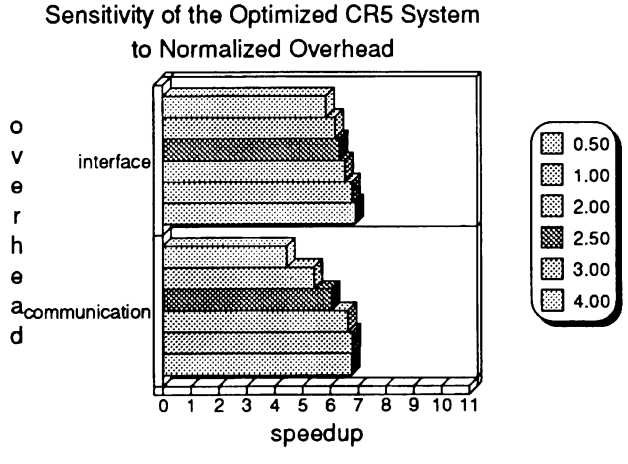

Figure 6. Sensitivity to Variations in Communication and Interface Times

4.2 Results from the CQn System

Results from the CQ3, CQ4 and CQ5 simulations are shown in Figure 7. As this queueing system was included to investigate model scalability, the following configurations were selected for experimentation. First, the (by now) standard four processor system; a system containing two "stats" (client and server states, respectively) and one RNS processor per queue, and one global priority queue processor; and finally, a system as above with four "stats" and one RNS per queue. Results from the four processor systems were consistent and uninteresting. From the second class of systems, worthwhile speedups (in the range of 6.7 to 8.1) were obtained, although with decreasing processor efficiency. This declining efficiency is probably due

to the use of only one priority queue processor. The most spectacular improvement was made in the third case, for the 22 processor CQ4 system. The average displayed (10.9) is conservative, for the system required a substantial time to stabilize. The speedups for the three blocks used to compute the graphed mean are 7.98, 13.18, and 13.34. Memory limitations precluded the simulation of the CQ5 class three (27 processor) system.

4.3 Results from the MMc System

The MMc simulator was used to test the effect of varying the load on one component (the priority queue processor). The algorithm for priority queue management was one developed by the author, with an observed asymptotic performance of O(sqrt n) for insertion, and O(1) for deletion. If all inserted entities have the same priority, both insertion and deletion are O(1). The results, shown in Figure 8, show a remarkable consistency. The event set ancilla was busy between 40 and 45 per cent of the time, regardless of event set size. Experiments with larger event set sizes were (again) precluded by memory limitations. Even with that observation, consistent speedups of 6 or more have not been yet reported for distributed

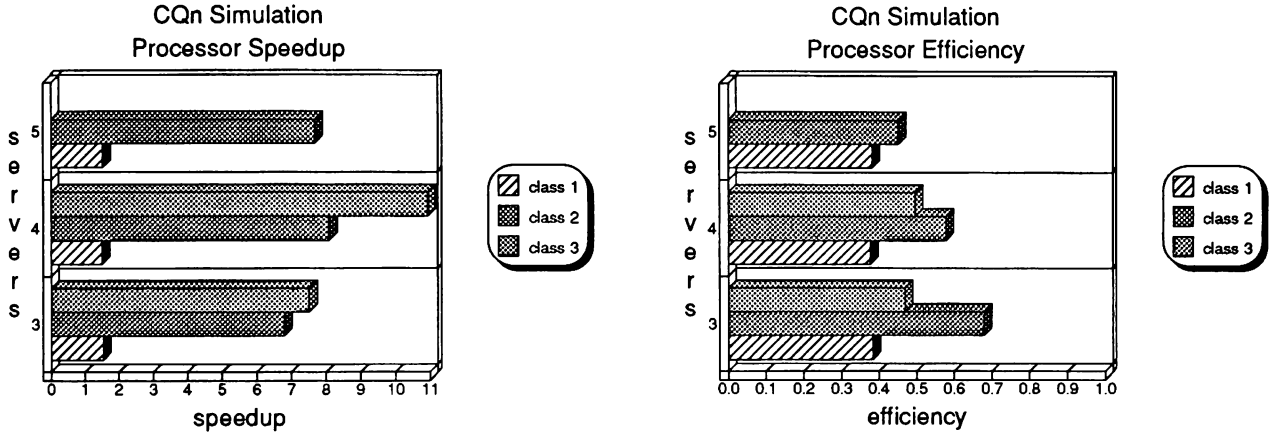

Figure 7. Performance of CQn Computer Systems

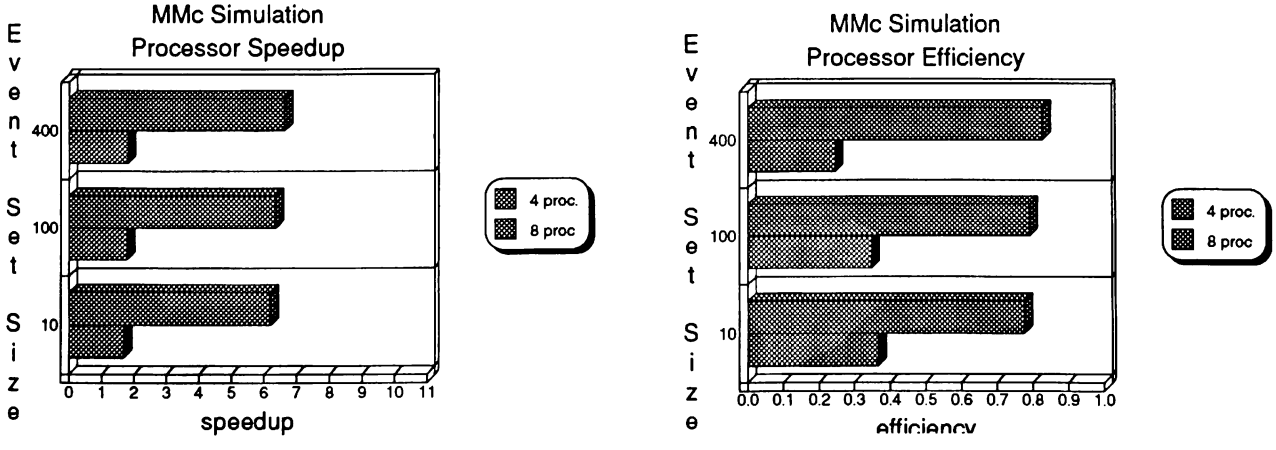

Figure 8. Performance of MMc Computer Systems

simulations of this kind of queue, so these results are encouraging. Further improvement, however, will probably require investigation into assigning fragments of instances to processors.

5 CONCLUSIONS

Environment Partitioned Simulation appears to provide an effective means of substantially reducing the run time of queueing simulation programs. The simulated results here presented for the CR5 and MMc systems are impressive, and should carry over into any queueing simulation composed of the three standard objects employed. In a simulation requiring additional primitive or higher level objects (for example, conveyor belts, in a shop floor simulation), the analysis and partitioning process could be extended to include these objects.

The strategy here employed would seem to be applicable to any discrete simulation model of low computational granularity. If a general simulation model can be partitioned into submodels, where the submodels have low internal computational granularity, but the macro-level system composed of the aggregated submodels has high computational granularity, then the model partitioned and environment partitioned approaches could be employed in fitting a distributed system to the complete model. The mechanism of assigning instances to ancillae here proposed is static. The problem of dynamic assignment and resultant load balancing appears interesting and challenging.

REFERENCES

Comfort, J. C., "The Simulation of a Master-Slave Event Set Processor", *Simulation,* 42,3 , pp. 117:124, March 1989

Jones, D.W., Chou, C., Rink, D., and Bruell, S.C., "Experience with Concurrent Simulation", *Proceedings of the 1989 Winter Simulation Conference,* Dec. 1989, IEEE Press, pp. 756-64

Lomow, G., Cleary, J., Unger, B., and West, D., "A Performance Study of Time Warp", *Distributed Simulation 1988*, SCS, pp. 50-55

RajaGopal, R., and Comfort, J.C., "Contrasting Distributed Simulation and Parallel Replication: a Case Study of a Queueing Simulation on a Network of Transputers", *Proceedings of the 1989 Winter Simulation Conference,* Dec. 1989, IEEE Press, pp. 746-54

Wagner, D.B., and Lazowska, E.D., "Parallel Simulation of Queueing Networks: Limitations and Potentials", Technical Report 88-09-05, Department of Computer Science, University of Washington, Seattle, 1988

AUTHOR'S BIOGRAPHY

John Craig Comfort received his Ph. D. in Computing and Information Science from Case Western Reserve University in 1974. He immediately joined the Computer Science Faculty of Florida International University, where he currently holds the rank of Professor. He has written many papers on simulation methodology and applications, specializing in Distributed Simulation. He has been an Associate Editor of Simulation, an ACM National Lecturer; has been Secretary, Vice President, and President of the Annual Simulation Symposium, and has held the Local Arrangements, Associate General, Associate Program, and Program Chairs of the Winter Simulation Conference. He occasionally designs puns.