Proceedings of the 1989 Winter Simulation Conference
E.A. MacNair, K.J. Musselman, P. Heidelberger (eds.)

CONTRASTING DISTRIBUTED SIMULATION WITH PARALLEL REPLICATION: A CASE STUDY OF A QUEUING SIMULATION ON A NETWORK OF TRANSPUTERS

R. Rajagopal
Late of the
School of Computer Science
Florida International University
Miami FL 33199

John Craig Comfort
School of Computer Science
Florida International University
Miami FL 33199
(305) 554-2015

1. ABSTRACT

As discrete event simulation programs become larger and more complex, the amount of computing power required for their execution is rapidly increasing. One way to achieve this power is by a employing a multiple processor network to run the simulation programs.

Two approaches to the problem of assigning tasks to processors are described--environment partitioning distributed simulation, in which the tasks required to perform a simulation are assigned to processors in the network; and parallel replication, in which copies of the simulation program are assigned to processors, and the results of their execution aggregated. A simulation of an M/M/c queuing system has been implemented on networks of two and three transputers, using each approach. Heidelberger's statistical efficiency and the stabilization time of the system are used as metrics. The parallel replications tended to stabilize faster, but the statistical efficiencies were not significantly different.

2. INTRODUCTION.

The availability of inexpensive microprocessors offers the possibility of significantly reducing run-times by applying more processors to the task of executing discrete event simulation programs. Determining strategies for distributing the tasks entailed by discrete event simulation over the set of available processors has been a subject of interest to the simulation community for the last several years. At least two primary approaches to distributed simulation have emerged: Model partitioning and Environment partitioning. In model partitioning, processing tasks associated with (related) functions of the system being modeled are assigned to individual processing elements; the inter-relationships between the processes is realized by communication among the processors through messages passed among them. In environment partitioning, those CPU-intensive functions required to support the simulation are analyzed and assigned a priori to processors ancillary to the main simulation processor [2].

The usual measures of effectiveness of distributed simulations are SPEED-UP and EFFICIENCY.If a simulation model executes in time t1 on a single processor and in time tp on a network of p identical processors, then the speed-up is defined as t1/tp, while efficiency is defined as the speed-up divided by p.

Heidelberger [2] suggests an approach, applicable to simulations of the steady state behavior of systems, different from both model and environment partitioning. Instead of having all the available processors cooperate on a single realization of the simulation, each processor runs a copy of the simulation model (PARALLEL REPLICATION), and results are aggregated. The only synchronization overhead is in loading the model into each processor at the beginning of runs, and averaging results together at the end of runs. This overhead is negligible resulting in near perfect efficiency. A theoretical model is formulated to contrast distributed simulation (model partitioning or environment partitioning) with parallel replications. Heidelberger defines an approach to simulation as having a higher STATISTICAL EFFICIENCY if it produces estimates with a smaller mean squared error for a given amount of computing resources.

3. THE PARALLEL ENVIRONMENT.

The goal of our research was to implement a prototype system employing both environment partitioned distributed simulation and parallel replications, to compare and contrast the two approaches, and to provide raw data to be used in the simulation of larger systems to investigate this question further. To implement a prototype requires a parallel computer environment on which to execute the simulation programs. For reasons of flexibility, cost, local availability, and prior experience, we chose to perform our research using the INMOS Transputer as the basis for the network and computing environment.

The Transputer is a new generation microprocessor specifically designed to serve as a network component. They are moderately fast processors with four bidirectional point to point links which may be connected to other processors in the Transputer family and local memory. T414 Transputers have a 32 bit multiplexed data and address bus. They execute approximately 7 million arithmetic / data move instructions per second. Floating point operations are done in software. The four links can operate at a (switchable) rate of 10 or 20 Million bits per second. Links, which provide point to point communication, have some advantages over often used bus communication scheme. There is no contention for the communication mechanism, regardless of number of Transputers in the network.

Communication bandwidth does not saturate as the size of the system increases [6], rather increases with number of Transputers in the system.

Transputers are available in a variety of configurations. Two such configurations (INMOS B004 and INMOS B003) are employed here. The INMOS B004 board contains one T414 Transputer, 2 megabytes of memory, and one INMOS C002 link adaptor. The B004 is an IBM PC XT or AT add-in board. The INMOS C002 interfaces the PC I/O channel to one of T414's link. This permits the Transputer network to use the IBM PC XT or AT both as an input/output device and as a file server. The INMOS B003 board contains four T414 Transputers each with 256 kilobytes of memory. Two links of each T414 Transputer are connected to its neighbor T414's, and the remaining two links are routed to an edge connector on the board, so that they are user assignable.

Transputers are reduced instruction set microprocessors, causing assembly language programming on them to be unusually tedious and annoying. OCCAM, a language of about the same level as C, is provided as the native language for Transputers. As described in [5], OCCAM provides a straightforward means of specifying parallelism, and for the creation and replication of processes. It also facilitates inter-process communication through user defined channels. OCCAM contains most of the control and data specification features present in more conventional programming languages. It views a program as a coordinated set of processes, communicating only through logical channels. Sharing resources (especially memory) among processes is not supported. Processes defined in OCCAM can be mapped to available processors by using the configuration utilities provided by the Transputer Development System. OCCAM allows programs containing a set of processes to be developed on a single processor. This program may then be recompiled and mapped on to any valid logical channel/hardware link, process/processor configuration. The multi-processor version of a program should work correctly if the uni-processor version did.

4. THE BENCHMARK PROGRAM.

The benchmark program throughout this investigation is a simulation model of an M/M/c queuing system, implemented in OCCAM. The M/M/c simulation model is realized both as a classical uni=process program, and as a set of multi-process programs using single and multiple processors.

The OCCAM language essentially forces an object oriented approach upon the programmer. The programs are composed of processes, each with its local storage, and connected to calling process by channels of relatively narrow bandwidth. To make efficient use of such a programming facility, it is necessary to implement all related processing activities within a process, using the channels to control their activation and to pass parametric information.

Two generic and two specific objects that could consume a significant amount of computing time were identified in the simulation model of the M/M/c queuing system. The two generic objects are the priority queue processor, which was instantiated as the event set (future events list) processor, the (customer) wait queue processor, and the (server) idle queue processor; and the random number generator, which was instantiated as the (customer) inter-arrival times generator and the (server) service times generator. These generic objects and their activation points are as described below.

```
PRIORITY QUEUE PROCESSOR
      { Instances:Event set,
                  Wait queue,
                  Idle queue }

   Initialize
      Initialize the local queue data
        structures.
      Initiate the appropriate
        priority queue process.

   Schedule ( IN identifier, priority )
      Create an EVENT NOTICE
         containing the identifier and
         its specified priority.
         Schedule this notice in the
         queue in order of increasing
         priority.
         NOTE -- Equal priorities are
         to be scheduled in the order
         in which they are received.

   Next ( OUT identifier, priority )
      Extract the notice with lowest
        priority from the queue.
        Return its identifier and
        priority.

   Terminate


RANDOM NUMBER GENERATOR
      {Instances:Inter-arrival
                 times,
                 Service times }

   Initialize ( IN type, seed, mean )
      Accept the type of the stream to
        be generated (negative
        exponential, uniform, or
        constant), its initial seed,
        and its mean, initialize local
        data structures. Initiate the
        appropriate random number
        generator process.

   Generate ( OUT random number )
      Return next value from specified
        stream.

   Terminate
```

Two specific objects are the statistical accumulator and the simulation administrator. They are as described below.

```
STATISTICAL ACCUMULATOR

    Initialize
       Set all entity times of creation
         and times of change to zero.
         Set all the facility times of
         creation and times of last
         change to zero.

    Accrue ( IN entity, state, time )
       Accumulate the sums and sums of
         squares necessary to compute
         first and second order time
         and utilization statistics.

    Aggregate ( OUT
                accumulated statistics )
       Using the raw statistical
         information accumulated by
         Accrue, compute the aggregated
         second order time and
         utilization statistics, and
         transmit them to the
         requesting process.

    Reset
       Reset the accumulators for time
         and utilization statistics.

    Terminate
```

```
SIMULATION ADMINISTRATOR

    Initialize
       Place all objects in simulation
         in their designated initial
         states.

    Run a block ( IN block-length;
                  OUT clock,
                      accumulated
                      statistics )
       Start the execution of next
         block. Run the block until the
         specified block length. Upon
         completion, transmit the
         aggregated statistics to the
         control program.

    Terminate
```

A connection diagram for these objects is shown in Figure 1. There are other objects identified in the queuing system simulation model, such as the report generator object and the object which queries user for queuing system parameters. They were not separated from the main simulation control procedure, either because there would be no apparent gain in execution time through parallelism, or because the objects would be activated for a relatively small amount of the total execution time.

The adaptive linked list algorithm (AL) is used to manage the priority queues, (event set, idle, wait). It employs an indexed doubly linked list to maintain queue entries in order of increasing priority. Algorithm AL has been shown to be more efficient than the more conventional non-indexed doubly linked list. No attempt is made to optimize the performance of any of the objects described above. The objects are designed with the goals of modularity, program clarity and information hiding in mind.

5. PARALLEL REPLICATION OF THE QUEUING SYSTEM SIMULATOR.

In order to implement parallel replication, the classical program of an M/M/c queuing system was considered. Two specific objects were extracted from that program; a simulation controller, and a simulation user interface. If parallel replication is done using n processors, then there will be n copies of simulation controller objects, each running on one processor. One simulation user interface object will be coordinating these simulation controller objects. A diagram of one three processor parallel replication is shown in Figure 2.

The simulation controller object contains and coordinates the previously mentioned objects -- the priority queue processor, the random number generator, the statistical accumulator, and the simulation

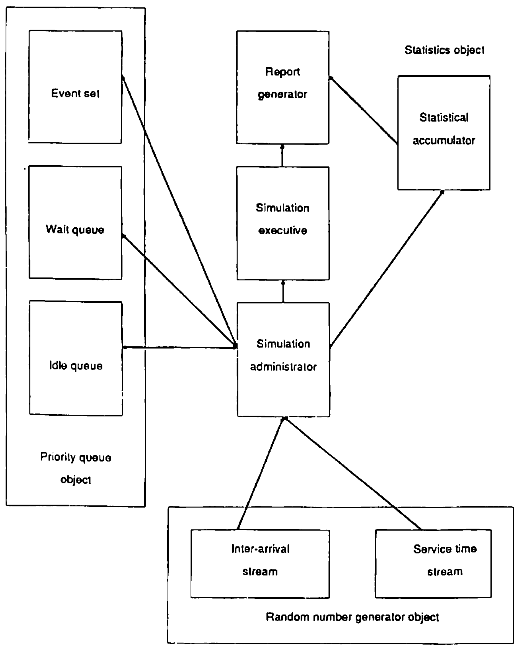


Figure 1. Objects in a M/M/c queueing system.

administrator. The simulation controller object receives all relevant user input data from the simulation user interface object, then initiates and runs the simulation for the specified number of blocks, and each block for the specified block length. At the end of each block, the simulation controller object sends the accumulated statistics to the simulation user interface object.

The simulation user interface object queries user for the parameters necessary to carry out the simulation. It sends relevant data to each copy of the simulation controller object. At the end of each execution block, each simulation controller object will send batch statistics, which will be aggregated by the simulation user interface object, then displayed to user.

The simulation controller and the simulation user interface object's activation points are as described below.

```
SIMULATION CONTROLLER

     Initialize ( IN queuing system
                          parameters,
                     block length,
                     number of blocks )
        Accept data needed to run the
          queuing system, of blocks to
          run, each block length.

     Generate ( OUT accumulated batch
                          statistics )
        Output accumulated statistics
          for each batch to simulation
          user interface object.

     Terminate


SIMULATION USER INTERFACE

     Query ()
        Query the user for queuing
          system parameters, number of
          simulation blocks, block
          length, data aggregation
          point.

     Initialize Simulation Controller
              ( OUT queuing system
                          parameters,
                     block length,
                     number of blocks )
        Send relevant data to the
          simulation controller.

     Aggregate ( IN accumulated
                      statistics )
        Receive accumulated statistics
          from all the simulation
          controllers. Aggregate them
          and display the aggregate
          statistics.

     Terminate.
```

6. Investigations.

First, an M/M/c queuing system simulation model was implemented in a classical manner on a Transputer system. This configuration is here referred to as the HOST. The objects described earlier (priority queue, random number generator, statistics accumulator, simulation administrator) were separated from the host and implemented in nine distinct configurations.

To maintain a high degree of compatibility among the different configurations, code for the main process and the object to be separated was not modified. Pairs of interface procedures were written. For example, consider the event set processing instance of the priority queue processor object. In any of the implementations, if the simulation administrator requires the scheduling of an event at a particular time it would execute a call to the appropriate (interface) procedure. In those implementations where the priority queue object has been separated, the interface procedure would then send, on a dedicated channel, the function code "schedule", and other parameters needed to schedule the event. As soon as the parameters are sent the interface procedure was free to continue. On the priority queue object side of the interface, the controlling procedure would initialize the data structure, and wait for channel input. When such input was received, it would be analyzed, and resulting local procedure invoked.

Of the nine different implementations mentioned above, three are realized on two Transputer configurations, four on three Transputer configurations and remaining two on four Transputer configurations. The two processor (host plus one network processor) implementations tested are called "3p", in which all three priority queue instances (priority queue processor object) are assigned to a network processor; "r", in which the random number generating object is assigned to a network processor; and "s", in which only the statistics object is so assigned.

The three processor (host plus two network processors) implementations are called "p+2p", in which the event set instance of the priority queue object is assigned to one network processor, wait and idle queue instance to another; "3p+r", in which the three priority queue instances are assigned to one processor in the network and the random number generator object to another; "3p+s", in which the three priority queue instances are assigned to one processor in the network, statistics accumulator object to another; and "r+s", in which the random number generator object is assigned to one network processor and the statistics object is assigned to another network processor.

The four processor (host plus three network processors) implementations are called "p+2p+s" and "3p+r+s". The same technique is followed here in assigning objects to network processors as in the two and three processor implementations.

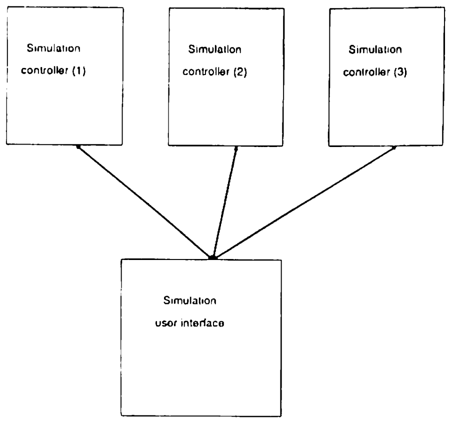

Simulation controller (1)
Simulation controller (2)
Simulation controller (3)
Simulation user interface

Figure 2. Three processor parallel replication

All the different implementations query the user for the same set of parameters. The parameters are:

- The mean and distribution of the inter-arrival times
- The mean and distribution of the service times
- The number of servers
- The number of blocks to be run
- The simulation time interval for each block (block length)
- The data aggregation block, after which data were accumulated
- A trace vector, used for debugging purposes.

The nine different implementations, described above, and the host program were run with the same set of parameter values. In this experiment, each batch was specified to run for a definite length of simulated time. The real time required to run each batch was noted for all runs. These run times were compared with the run time of the host configuration to find the speed-up and efficiencies of different implementations.

In the next investigation, the simulation controller and simulation user interface objects were defined for the same simulation model. Two configurations (two and three processor) of parallel replications were implemented. These will be referred to as 2-parallel and 3-parallel replications.

In the two (host processor plus two network processors) and three (host processor plus three network processors) processor implementations, one simulation user interface object was assigned to the host processor, and two and three simulation controller objects were assigned to the network processors. The user interface object was initiated, and aggregated the block results produced by the simulation controller objects.

These two implementations queried the user for the same set of parameters (which are listed above) as did the environment partitioned distributed simulation models.

The two and three processor replications were compared with the best two ("s") and three ("r+s") processor environment partitioned models. For this comparison, the block length parameter was changed from a simulated to a real time value, as suggested by Heidelberger.

For every run of an n-parallel replications model, there were performed n runs of an equivalent environment partitioned model. For example, if the three processor replication model was simulated for n batches using three different random seed pairs (one for inter-arrival time distribution, other for service time distribution), then a three processor environment partitioned model should be run three times, each time with one of the random seed pairs used for the three processor replications, and each run was for n blocks.

Each batch run produces important statistical quantities: the mean and standard deviation of the server utilization and the mean and standard deviation of the wait time of entities waiting in the queue. From these quantities, a ninety five per cent confidence interval was computed for both server-utilization and wait time.

Theoretical mean server-utilization and mean wait time were computed for an ideal M/M/c queuing system from formulas provided in [16]. During the run, ninety five per cent confidence intervals were computed on the statistics generated by simulation models and these confidence intervals contained the theoretical means, after the models were simulated beyond transient state.

To compare the two different approaches -- environment partitioning and parallel replication -- mean squared errors were computed for the statistical quantities mentioned above. The approach which produced a lesser mean squared error is considered to be statistically more efficient [3]. Mean squared error is a criterion that combines the variance and the bias associated with a statistic of interest [4]. Bias is the difference between simulated mean and theoretical mean.

7. RESULTS.

For the environment partitioned distributed simulation experiments, there are two independent variables: event set size and object/network configuration. Each run was made for a simulation of a queuing system very close to saturation (0.99 load factor), so that the event set size is effectively equal to the number of servers. The parameter values used were, for n servers, a mean inter-arrival time of 10 and service time of 10*n. The negative exponential distribution was used for both inter-arrival and service times. FIFO discipline was used for (server) idle queues and (entity) wait queues. Block length of each run was 10000 units of simulated time, thus yielding roughly 1000 clients per block. Each configuration was run for eight such blocks at a time. The run times computed was the average of run times from third block to eighth block. Runs were conducted for event set sizes of 25, 50, 100, 200, 400, 600, 800, 1000, and 1200.

Speed-up factors and processor efficiencies for different configurations were computed from the observed run times. Speed-up factors ranged from 0.96 to 1.3 for two processor systems, 0.96 to 1.6 for three processor systems, and 1.2 to 1.6 for four processor systems. Speed-up increased with event set size for those systems in which the priority queue object was assigned to a network processor. For event set size less than 400, the "r+s" configuration showed the best speed-up of all configurations tested. For event set sizes beyond 400, the "r+s+3p" configuration showed better speed-up. In case of the "r+s+3p" configuration, the speed-up was nearly independent of event set size, showing a total variation of four per cent when the event set size varied from 25 to 1200. The lowest processor efficiency observed among all configurations was 0.31 for "p+2p+s", and the highest was 0.64 for "s".

The results from the experiments described above are shown in Tables 1a through 1c, and graphically in Figures 3 through 5. The run times, speed-up, and processor efficiencies are tabulated for each configuration. Separate graphs are plotted for two, three and four processor configurations.

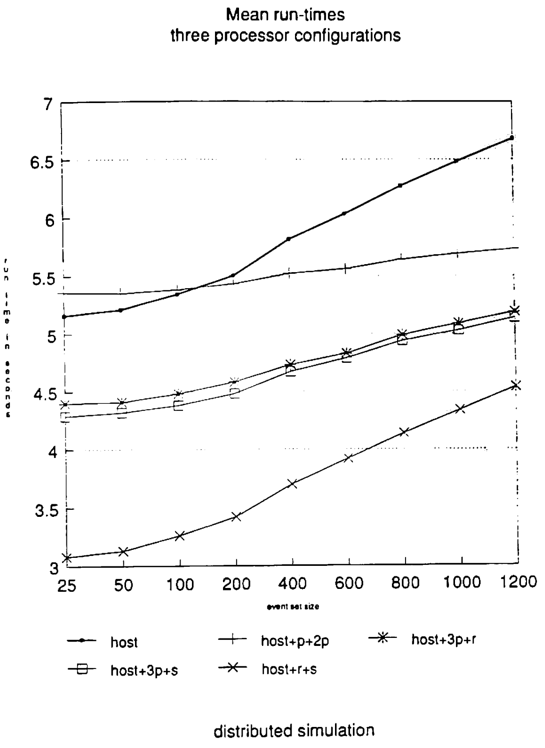


FIGURE 4.

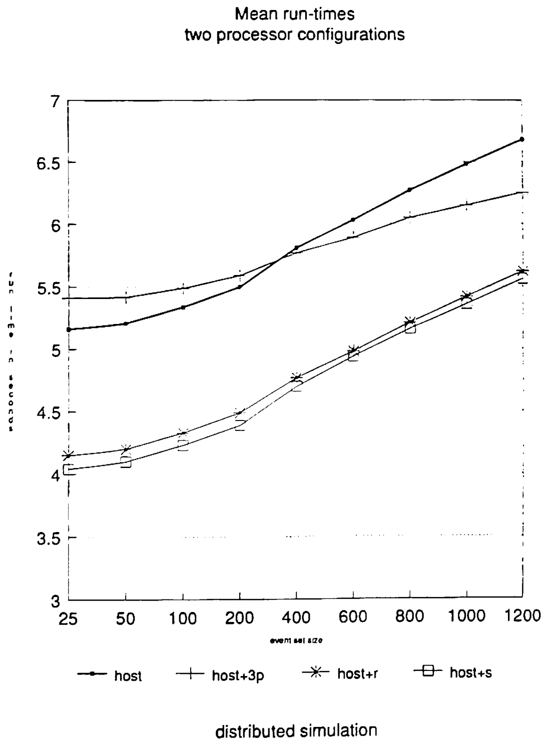


FIGURE 3.

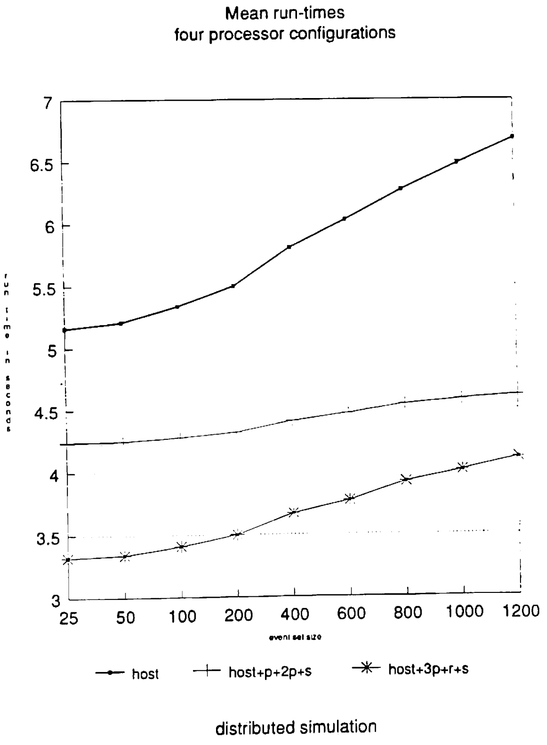


FIGURE 5.

In the next experiment, 2- and 3-parallel replication configurations were contrasted against the "s" and "r+s" environment partitioned configurations. All configurations were run for 80 blocks of 5 seconds (real time) each. They were run with three different load factors (if mean inter-arrival time is 10, mean service time is 180, number of servers is 20 then load factor is 0.9) i.e. 0.9, 0.75, and 0.5. The simulation runs were conducted for three different number of server values -- 20, 100, and 200. (The event set size was limited by the relatively small amount of memory (32K Bytes) to each network processor). For all the simulation runs the data aggregation was started beginning at block six. These configurations were run with different random seed pairs (inter-arrival time, service time).

As mentioned above, the data gathered in these experiments were mean and standard deviation of server-utilization, mean and standard deviation of wait time. From these factors ninety five per cent confidence interval was computed and compared with the theoretical values and mean squared errors for the respective configurations were computed.

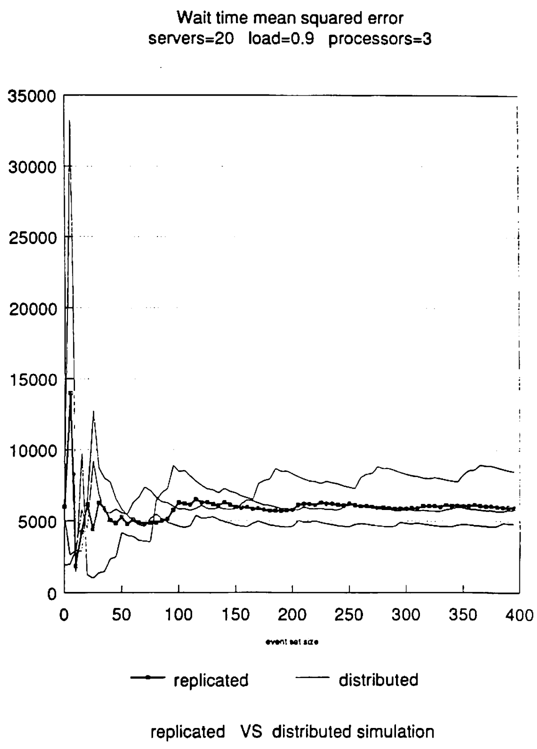
Parallel replication configurations have near perfect speed-up because there is no overhead involved, except for the initial loading of the processors and at the end of each block collecting the data. In case of environment partitioned configurations, the highest speed-up observed was 1.6.

In general, parallel replication configurations reached steady state earlier compared to their counterpart environment partitioned configurations. But surprisingly, the mean squared errors produced by both the approaches in all configurations were not significantly different from each other.

Figures 6 through 8 contain selected graphs for mean squared error of server-utilization and mean squared error of wait time for the 3-parallel replication and the "r+s" configuration. The load factors and number of servers (event set size) used to achieve these results are listed. Comparison of the two processor configurations produce results similar to that of the three processor configurations, so they are not shown. Server-utilization does not show much variation under heavy load conditions so graphs were not presented for that attribute.

8. CONCLUSIONS.

Environment partitioned simulation has shown promise as an effective way to reduce the cost of performing large, complex, time consuming simulations. The Transputer shows great potential for use in distributed simulation. However, the limited physical links (four per processor) impose relatively



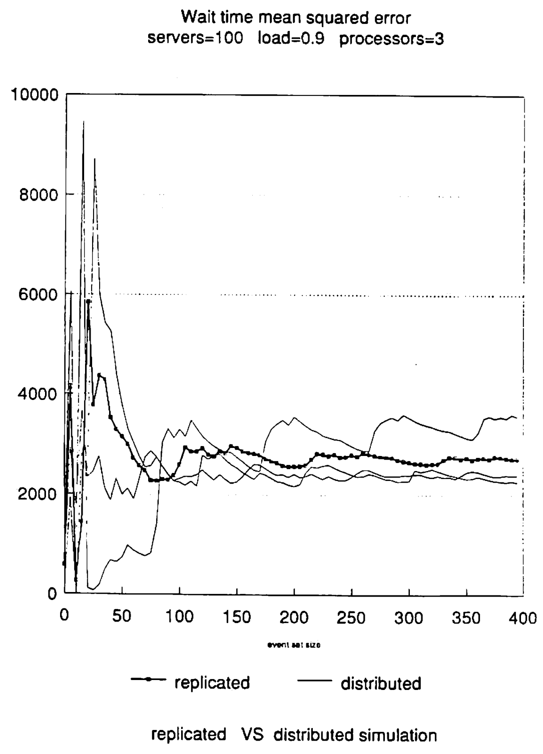


FIGURE 6.

low connectivity. This limitation will be overcome by the addition of link multiplexors to the system. These link multiplexors provide thirty two virtual links for each processor which can be switched by software. These multiplexors will be essential in creating environment partitioned simulations for larger systems.

Statistics produced by parallel replication configurations stabilize sooner than those of their corresponding environment partitioned configurations. After comparing the environment partitioning with parallel replication it cannot be decisively concluded which approach is statistically more efficient. This may be due to the fact that a small number of processors were involved.

REFERENCES.

[1] Heidelberger, P., "Statistical Analysis of Parallel Simulations", Proceedings of the 1986 Winter Simulation Conference, December 1986, pp 290-295.

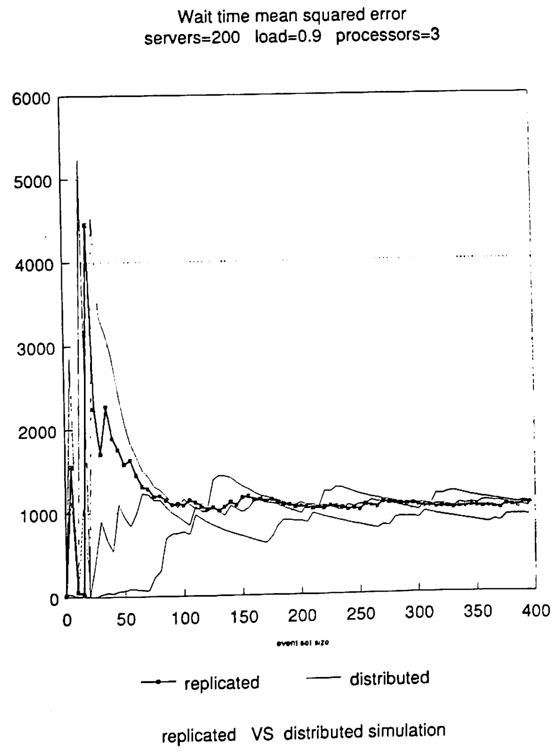
[2] Comfort, J.C. and Raja Gopal, R., "Environment Partitioned Distributed Simulation with Transputers", Proceedings of the 1988 Winter Simulation Conference, Feb 1988, pp 103-108.

[3] Pritsker, A., Introduction to Simulation and Slam II, New York: John Wiley and Sons, 1984.

[4] Allen, A., Probability, Statistics, and Queuing Theory, New York: Academic Press, 1978.

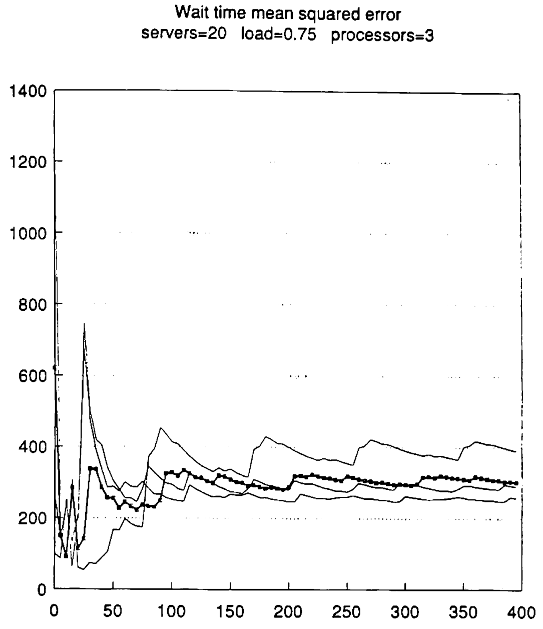
[5] Pountain, D., A Tutorial Introduction to Occam Programming, Inmos, 1986.

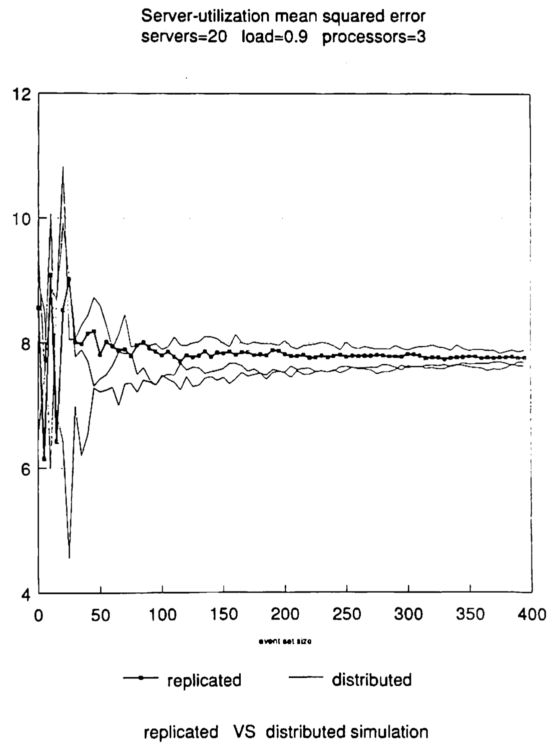
[6] Transputer Development System Manual, INMOS, 1987.



replicated VS distributed simulation

FIGURE 7.





replicated VS distributed simulation

FIGURE 8.

Mean Run Times (in seconds) for the M/M/c Simulation Program
in Different Configurations.

Event set size	Network Configurations									
	Host	3P	R	S	P+2P	3P+R	3P+S	R+S	P+2P+S	3P+R+S
25	5.16	5.41	4.15	4.04	5.36	4.40	4.29	3.08	4.24	3.32
50	5.21	5.42	4.20	4.10	5.35	4.41	4.32	3.13	4.25	3.34
100	5.34	5.49	4.33	4.23	5.38	4.48	4.38	3.26	4.28	3.41
200	5.50	5.59	4.49	4.39	5.43	4.58	4.48	3.42	4.32	3.50
400	5.81	5.77	4.77	4.70	5.52	4.73	4.67	3.70	4.41	3.67
600	6.03	5.89	4.98	4.94	5.56	4.83	4.79	3.92	4.47	3.77
800	6.27	6.05	5.21	5.16	5.64	4.99	4.94	4.14	4.54	3.92
1000	6.48	6.15	5.42	5.36	5.69	5.09	5.03	4.34	4.58	4.01
1200	6.68	6.25	5.62	5.56	5.73	5.19	5.14	4.54	4.61	4.11

Table 1a. Results of Environment Partitioned Distributed
Simulation of an M/M/c Queuing System

Event set size	Network Configurations									
	Host	3P	R	S	P+2P	3P+R	3P+S	R+S	P+2P+S	3P+R+S
25	1.00	0.96	1.24	1.28	0.96	1.20	1.20	1.68	1.22	1.56
50	1.00	0.96	1.24	1.27	0.97	1.18	1.21	1.66	1.23	1.56
100	1.00	0.97	1.23	1.26	0.99	1.19	1.22	1.64	1.25	1.57
200	1.00	0.99	1.23	1.25	1.01	1.20	1.23	1.61	1.27	1.57
400	1.00	1.01	1.22	1.24	1.05	1.23	1.24	1.57	1.32	1.58
600	1.00	1.02	1.21	1.22	1.08	1.25	1.26	1.54	1.35	1.60
800	1.00	1.04	1.20	1.21	1.11	1.26	1.27	1.51	1.38	1.60
1000	1.00	1.06	1.20	1.21	1.14	1.27	1.29	1.49	1.42	1.62
1200	1.00	1.07	1.19	1.20	1.17	1.29	1.30	1.47	1.45	1.62

Table 1b. Processor Speed-up for the M/M/c Simulation Program
in Different Configurations.

Event set size	Network Configurations									
	Host	3P	R	S	P+2P	3P+R	3P+S	R+S	P+2P+S	3P+R+S
25	1.00	0.48	0.62	0.64	0.32	0.39	0.40	0.56	0.30	0.39
50	1.00	0.48	0.62	0.64	0.32	0.39	0.40	0.56	0.31	0.39
100	1.00	0.49	0.62	0.63	0.33	0.40	0.41	0.55	0.31	0.39
200	1.00	0.49	0.61	0.63	0.34	0.40	0.41	0.54	0.32	0.39
400	1.00	0.50	0.61	0.62	0.35	0.41	0.42	0.52	0.33	0.40
600	1.00	0.51	0.61	0.61	0.36	0.42	0.42	0.51	0.34	0.40
800	1.00	0.52	0.60	0.61	0.37	0.42	0.42	0.51	0.35	0.40
1000	1.00	0.53	0.60	0.60	0.38	0.42	0.43	0.50	0.35	0.40
1200	1.00	0.53	0.59	0.60	0.39	0.43	0.43	0.49	0.36	0.41

Table 1c. Processor Efficiency for the M/M/c Simulation Program
in Different Configurations.

R. Rajagopal received his Master of Science degree from the School of Computer Science of Florida International University in 1989. He has been employed by Logica Technical Systems and the IBM Corporation.

John Craig Comfort received his Ph.D. degree in 1974 from Case Western Reserve University. He then joined the faculty of the School of Computer Science of Florida International University, where he currently holds the rank of Professor. He has been active in Simulation since 1976, having presented many papers at simulation conferences. He has been the President of the Annual Simulation Symposium, and was last year's (hooray!) Program Chairperson for the Winter Simulation Conference.

He may be contacted at the following address:

John C. Comfort
School of Computer Science
Florida International University
Miami FL 33199
(305)554-2015
COMFORT@SERVAX.BITNET