AN EXPERIMENT IN MICROPROCESSOR-BASED DISTRIBUTED DIGITAL SIMULATION

Dana L. Wyatt
Instructor
Dept. of Computer Science
Texas A&M University
College Station, TX 77843

Sallie Sheppard
Associate Professor
Dept. of Computer Science
Texas A&M University
College Station, TX 77843

Robert E. Young
Associate Professor
Dept. of Industrial Engineering
Texas A&M University
College Station, TX 77843

This paper discusses the design of a distributed simulation system which will utilize off-the-shelf microprocessors in its implementation. Alternative approaches to the assignment of simulation functions and processes are presented. A project currently underway at Texas A&M University is described which considers the impact of distributed architectures on the design of simulation language support systems. The emphasis in this research project is to produce an operational prototype which can be used to establish the feasibility and utility of distributed simulation.

1. INTRODUCTION

Simulation has long been recognized as a useful tool in the solution of many types of problems not easily solved using analytic techniques. However, one of the primary disadvantages of simulation is the sizable amount of computer time and mainframe hardware typically required to execute complex models. This problem is especially acute in applications where a real-time solution is desired.

The traditional approach for applications where time is crucial has been to utilize large mainframes because of their speed. However, these mainframes are very expensive. An alternative approach which has been advocated as a means of economically increasing computing power is to concurrently employ multiple smaller processors in the solution of a single problem (Scherr 1978). Such an approach offers not only initial savings in hardware cost, but also cheaper maintenance costs since each modular component is simpler and can be more easily repaired when required.

The parallel use of multiple processors as a means of increasing computing power has not yet become widespread, primarily because such computer architectures require special algorithms to take advantage of the hardware. Additionally, not all application areas are amenable to such parallel algorithms. However, simulation applications typically utilize unique structures which allow task separation and desynchronization, basic prerequisites for parallel processing. Research reported in the literature has indicated that this multiprocessor approach has been successfully followed in continuous simulation systems with some complete systems in existance (Halin, et al 1980, O'Grady 1979).

A project is currently underway at Texas A&M University funded by a grant from the National Science Foundation in which a microprocessor-based distributed digital simulation system is being developed (Sheppard, Phillips and Young 1982). The emphasis in this research is to produce an operational prototype which can be used to establish the feasibility and utility of distributed simulation. The research objectives are:

1. To conceptually design a simulation language and support environment based upon distributed processing,

2. To implement these constructs through the construction of an executable simulation system, and

This material is based upon work supported in part by the National Science Foundation under Grant No. ECS-8215550

3. To evaluate the feasibility and utility of distributed simulation.

Research related to a simulation support development environment developed is described in a companion paper (Reese and Sheppard 1983).

2. DISTRIBUTED SIMULATION

A distributed simulation system could be defined as a distributed computer system upon which simulation applications are implemented and executed in a parallel manner. Such a system provides simulation facilities in a distributed computing environment.

The term distributed computing system is one over which considerable controversy has emerged. Researchers are unable to agree on a complete, concise definition (Eckhouse and Stankovic 1978). However, a working definition of the term specifies that "a distributed system consists of a collection of distinct processors which are spatially separated, and which communicate with one another by exchanging messages" (Lamport 1978).

Two approaches are possible to the implementation of discrete simulation in a distributed manner, with the differences based on which simulation functions are distributed to the available processors. Distribution is possible by simulation support function and by model function.

2.1 Separation By Simulation Support Function

Current simulation languages are executed sequentially because they are written in a standard environment designed for sequential execution. However, a task analysis of typical simulation applications indicates many activities can be identified whose execution does not require synchronization with other simulation activities. Such activities can be classified as either anticipatory in nature or non-interactive.

An "anticipatory activity" is an activity whose need can be anticipated and performed in advance. An example is random number generation. A "non-interactive activity" is one whose performance, once initiated, has no immediate impact upon the other activities. An example would be inserting an entity into a queue.

Identification and examination of anticipatory and non-interactive actitivies allows them to be grouped into specific categories. Within each category exist activities which are interdependent upon one another. In a multiprocessor environment, each activity category becomes a candidate to be assigned its own processor. Such a system would run asynchronously and in parallel with one processor dedicated to system management while the other processors were responsible for handling the various activity categories.

Discrete simulation application processing typically involves such activities as random number generation, data input/output, statistics collection and processing, filing, etc. Thus, by distributing the simulation support functions over the available processors while executing the model functions in the traditional sequential manner (see Figure 1), it is possible to gain some advantages of parallel processing without the significant problems of synchronization and deadlock protection found when using the alternate approach.

2.2 Separation by Model Function

Real systems for which models are typically constructed are inherently parallel with asynchronous operation of the components. Distribution of the model functions based upon this inherent parallelism of the real system involves the parallel execution of the process (event) routines as described by the model (see Figure 2). Those processes which occur in parallel in the real system could be simulated in parallel in the distributed simulation system.

This approach is esthetically pleasing because it allows the physical components of the world which occur in parallel to be truly executed as such in the simulation. With an appropriately designed langauge, the model builder could use the observed parallelism in creating the model rather than being forced to mentally reconfigure the problem to match the sequential constraints of, for example, event-oriented languages. The result would be similar to a process-oriented language such as SIMULA (Dahl, Myhrhaug and Nygaard 1968), which could actually execute in parallel.

This approach, however, is considerably more difficult to implement as there are at least three

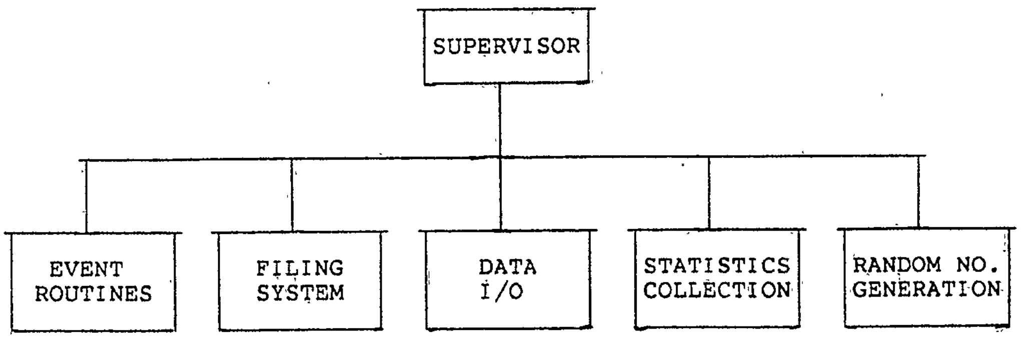

Figure 1: Simulation Distributed by Simulation Support Function

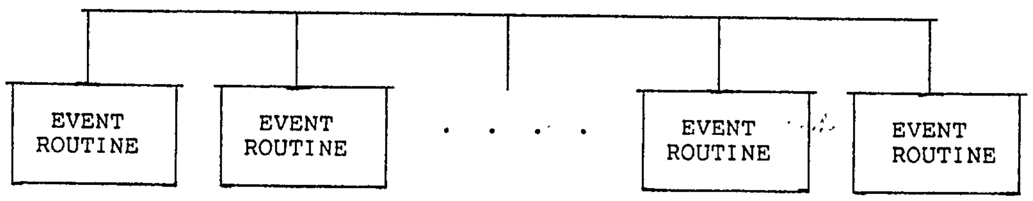

Figure 2: Simulation Distributed by Model Function

new problems introduced over those encountered in the hierarchical approach. First, the simulation language itself will require additional facilities not typically included in present simulation languages, in order to allow the model builder access to and control of the parallel capabilities.

Secondly, resource allocation becomes a problem. Although logically one would expect each process to be assigned to a separate processor, some realistic constraints must be imposed based on the number of actual processors available. Ideally, these constraints should be transparent to the model builder.

The third problem involves the synchronization and deadlock protection among the processes. With each routine assigned to its own processor and all processors running in parallel, specific actions must be taken to synchronize their activities as required by the real world situations being modeled. Without such synchronization, some processes may "get ahead" of others and incorrectly model interdependencies in the system. Deadlock may occur if two or more processes are waiting on each other with forward progress of all being blocked. Although a hierarchical configuration of the processors solves most of the problems of synchronization and deadlock protection, a configuration without a central supervisor is needed to maximize the distribution and desynchronization of processes. Such an arrangement, however, introduces problems of synchronization and the possibility of system deadlock complicates interprocess communication and data sharing.

3. PREVIOUS RESEARCH

Research in the area of distributed discrete simulation has only been reported in the past few years. Two primary research efforts were reported by the University of Texas at Austin (Chandy, Holmes and Misra 1979, Chandy and Misra 1981), and the University of Waterloo (Peacock, Manning and Wong 1979, Peacock, Wong and Manning 1980). These two efforts represented part of a joint program whose goal was to solve parallel problems using message-switched networks. Additionally, there have been other independent research projects in distributed discrete simulation (Bryant 1979, Comfort 1982, Reynolds 1982, Wittie 1978).

With one exception, these research efforts have dealt primarily with distributed simulation implemented with the distribution of tasks to the processors made through model function. Because of this, research efforts to date have dealt mostly with such topics as the location and control features of the simulation clock, communication facilities and protocols between processes, synchronization and control of the processes, resource allocation among the processors, and data sharing capabilities.

The only research effort found which deals with distributed discrete simulation implemented in the hierarchical manner of Figure 1 is the work of Comfort (1982). He developed a system using a PDP-11 as the principal processor and distributed the event set processing to a M68000 microprocessor. He has been fairly successful, programming the PDP-11 in FORTRAN-IV PLUS and the M68000 in assembly language. The emphasis of his research has been in the performance of the system and how it varies with different interconnection and buffering strategies rather than on simulation language concerns.

The performance degradation due to dynamic deadlock detection and prevention inherent in the distribution by model function is severe enough to warrant many investigations into alternative approaches to distributed discrete simulation. Multiprocessor systems with architectures using a hierarchical structure have been suggested in the literature (Elizas 1979, Enslow 1977) and are being investigated in the system being developed at Texas A&M University.

3.1 Simulation Languages

There are no parallel or distributed languages commercially available which are designed specifically for simulation. Quasi-parallel simulation languages do exist, in the form of process-oriented discrete languages. One of the first of these was SIMULA. SIMULA allows the programmer to specify the system in terms of a number of concurrent, synchronized processes. However, these processes are interleaved so that they can be executed in a sequential manner. Since SIMULA was introduced, other process-oriented languages have appeared.

The popularity, availability, and economic advantages of microprocessors has recently had an impact on simulation languages. Byrant (1981a) has been exploring the use of microprocessors as a host computer for a simulation language. He has developed Micro-SIMPAS, a Pascal-based simulation language for microcomputers. Although he found that the microprocessor software was sufficiently powerful to support simulation, the microprocessor hardware available at the time was not fast enough to make simulation feasible. Distribution of the workload over several microprocessors should advantageously affect the performance of simulation on microprocessors.

3.2 Distributed Languages

As the construction of general purpose systems from smaller processors has become widespread, many of the conventional programming languages can no longer be easily adapted to run on these multiprocessor systems (Silberschatz 1980). As a result, a new group of languages is beginning to emerge. These new languages may be classified into two types according to the type of communication facilities they use: those that are based on buffered messages (e.g. PLITS (Feldman 1979) and E-Clu (Liskov 1979)) and those based on synchronous unbuffered messages (e.g. DP (Brinch Hansen 1978), CSP (Hoare 1978), and Ada[R] (Notkin 1980)). Each of these is specifically designed to be run on a multiprocessor computer system. However, the features typically included in simulation languages are not part of any of the distributed languages.

4. DISTRIBUTED SIMULATION SYSTEM DESIGN

The design of the distributed simulation system currently being developed at Texas A&M University is based on the hierarchical architecture of Figure 1 in which the simulation support functions are distributed to the available processors. Two operational prototypes are being constructed based on different existing simulation languages: SIMPAS (Bryant 1981b) and GASP IV (Pritsker and Young 1975). After experience and insight has been gained from these prototypes, the alternative approach, distribution based on model function, will be considered. The work presented in the remainder of this paper concerns the prototype based on SIMPAS.

SIMPAS is a strongly typed, event-oriented, discrete simulation language based on Pascal. It is implemented as a preprocessor which accepts as input an extended version of Pascal and produces a standard Pascal program as output. This enhances the transportability of the language.

The extensions to Pascal which SIMPAS incorporates are similar to those of SIMSCRIPT. A simulation program written in SIMPAS can be divided into seven basic parts: global label declarations, global constant declarations, global type declarations, global variable declarations, procedure and event declarations, and the main procedure.

A SIMSCRIPT-like entity can be created in SIMPAS using the following declarations:

```
type
  ship = queue member
     ship_id              :  integer;
     arrival_time         :  real;
     unloading_time       :  real;
     loading_time         :  real;
  end;

  ship_queue = queue of ship;
```

```
var
  tanker            : ship;
  freighter         : ship;

  harbor_line       : ship_queue;
```

To refer to the particular attributes of an entity, one would use the Pascal dot notation referring to fields of a record:

```
     tanker.arrival_time
           or
     freighter.loading_time
```

The SIMPAS extensions to Pascal consist of "new" statements which have specific simultion functions. Samples are given in Table 1.

STATEMENT	DESCRIPTION
INCLUDE	to include library pseudorandom number generation routines
START SIMULATION	to begin execution of the events
SCHEDULE	to create events and insert them into the event set
CANCEL	to remove a named event from the event set
DESTROY	to dispose of a previously canceled event
RESCHEDULE	to reschedule the current event at a later time
INSERT	to insert entities into a queue
REMOVE	to remove entities from a queue
FORALL	to scan the members of a queue

Table 1: Sample SIMPAS Simulation Statements

In addition to these statements, a new type of procedure is available to indicate the actions which occur during the execution of an event. An event declaration has exactly the same form as a Pascal procedure declaration, except that the word "EVENT" replaces "PROCEDURE" in the declaration. The only restriction on event declarations is that they may not be delcared local to a procedure or another event in order to allow them to be accessable from the event set scanning routines.

The first phase of this research project involved implementing SIMPAS on a Texas Instruments 990/12 minicomputer. The execution of a SIMPAS program is a two step process. First, the SIMPAS program must be processed by a preprocessor. The SIMPAS

[R]Ada is a registered trademark of the U.S. Department of Defense.

preprocessor is divided into two passes, with the first being a parser, of sorts, and the second expanding the SIMPAS statements into Pascal code. The Pascal output of the preprocessor must then be compiled. Only then can the object program be run. Even though this translation process is slow, the selection of SIMPAS proved valuable because of the ease in which its preprocessor is being modified to produce a Pascal program to be executed in a distributed manner.

4.1 Emulation of the Distributed Simulation System

The TI 990/12 minicomputer system being used in this research project is a 16-bit minicomputer. It is being used both as a software development system and as an emulator for the prototype distributed simulation system. The TI 990/12 allows multitasking of up to 8 64K byte processes, which corresponds exactly with the actual microprocessor network targeted as the prototype hardware. Communication among the tasks on the TI 990/12, because of constraints imposed by the operating system, is through a global common data area using semaphores for synchronization.

An analysis of the SIMPAS lanaguage and preprocessor has produced an initial design of the distributed simulation system consisting of five processes (or tasks) as shown in Figure 3.

The standard version of the SIMPAS preprocessor produces a Pascal program which is designed to be executed in a sequential manner. In order to generate a program designed to be executed in a multitasking manner, the SIMPAS preprocessor is being modified. Pass 2 of the preprocessor, in which SIMPAS statements are expanded to generate Pascal code to accomplish the specified task, is being rewritten so that "calls" to the appropriate task are generated.

For example, SIMPAS expands the INSERT statement to the appropriate set of Pascal statements to allocate and link the specified entity into the appropriate queue. These statements are being replaced by a "call" to the filing system task. Similarily, REMOVE will now invoke the filing system task and force the event routine to wait until the entity is available.

Random number generation routines running in parallel with the other routines will keep a supply of numbers available so that the event routine should not have to wait for a number. As a result, the Pascal program no longer "generates" random numbers, per se, but rather it accesses a predefined data area which contains a large supply of random numbers. Only if no random numbers are available would it be required to wait for the generation of a number.

Statistics collection is being implemented as a "passive task" which "watches" for specific changes in the data area. It then collects statistics similar to those of SIMSCRIPT. User-defined statistics collection will still be executed in the event routine task.

It is the responsibility of the supervisor task to initiate each of the sub-tasks, providing semaphores to control the message passing, and monitoring an error flag area so as to intercept and attempt to handle any errors which might arise during the execution of the simulation program.

4.2 Implementation on a Microprocessor Network

The implementation of the prototype distributed simulation system was originally planned for a network of 8 TI 990 microprocessors connected via a multidrop network. However, since the multidrop network is very slow when passing the large quantities of data, another architecture is seriously being considered. Negotiations are underway to acquire several Motorola 68000 based microprocessors to be used in place of the TI 990 microprocessors.

5. CURRENT STATUS AND ANTICIPATED BENEFITS OF RESEARCH

The first phase of this project involves the selection of a simulation language and the initial design of the distributed simulation system and has been completed. As of this writing, work has recently begun on phase two. When completed, this phase will produce a prototype discrete simulation language implemented on a minicomputer in such a manner that it emulates a distributed discrete

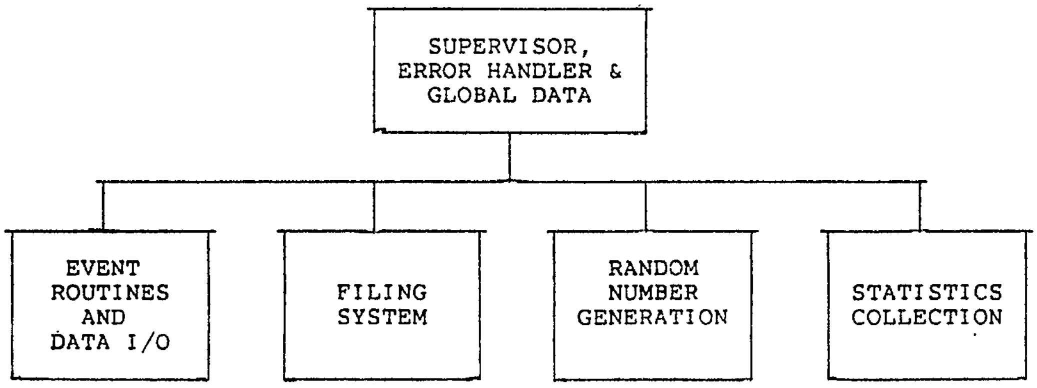
SUPERVISOR, ERROR HANDLER & GLOBAL DATA
EVENT ROUTINES AND DATA I/O
FILING SYSTEM
RANDOM NUMBER GENERATION
STATISTICS COLLECTION

Figure 3: A SIMPAS-based Distributed Discrete Simulation System

simulation system. Observations on this prototype will then be made as to the amount and type of interprocess communication which occurs in this method of distributing simulation support functions, allowing recommendations to be made concerning multi-microprocessor architecture and hardware requirements to be used for implementation. Additionally, alternative approaches to the hierarchical architecture selected for this project will be examined to determine their practicality and the impact they might have on the simulation language design. Once these tasks have been completed, the implementation of the distributed simulation system will begin on a microprocessor network.

Although other researchers have designed distributed discrete simulation systems, this research differs in three important ways:

1. Most designs for distributed discrete simulation call for implementation through the distribution of the model functions, introducing severe problems of synchronization and deadlock protection. This research explores an alternative hierarchical approach that contains no major synchronization or deadlock protection problems.

2. Most research in distributed discrete simulation has been in the physical design of the system, with emphasis placed on deadlock detection mechanisms and performance issues. This research shifts the emphasis to the software aspect, and in particular, the implementation requirements of a simulation language which uses the distributed architecture.

3. This research will provide statistics and observations made on the amount and type of interprocess communication found in modularized simulation applications.

When completed, this research will determine the requirement specifications of a multi-microprocessor architecture for implementation of a hierarchical approach to simulation and provide an operational prototype for determining the feasibility and utility of distributed discrete simulation implemented in a hierarchical manner.

REFERENCES

Brinch Hansen P (1978), Distributed Processes: A Concurrent Programming Concept, Comm. ACM, Vol. 21, No. 11, pp. 934-941.

Bryant RE (1979), Simulation on a Distributed System, 1979 Distributed Computing Systems Conf., pp. 544-552.

Bryant RM (1981a), Micro-SIMPAS: A Microprocessor-based Simulation Language, Proc. 14th Annual Simulation Symp.

Bryant RM (1981b), SIMPAS Users Manual, Dept. of Computer Science and Academic Computing Center, University of Wisconsin - Madison, Madison, WI.

Chandy KM and Misra J (1981), Asynchronous Distributed Simulation via a Sequence of Parallel Computations, Comm. ACM, Vol. 24, No. 11, pp. 198-206.

Chandy KM, Holmes V and Misra J (1979), Distributed Simulation of Networks, Computer Networks, Vol. 3, No. 1, pp. 105-113.

Comfort JC (1982), The Design of a Multi-Microprocessor Based Simulation Computer - I, Proc. 15th Annual Simulation Symp., pp. 45-53.

Dahl OJ, Myhraug B and Nygaard K (1968), SIMULA 67: Common Base Language, Norwegian Computing Center, Forskringsveien 1B, Olso 3, Norway.

Eckhouse RH and Stankovic JA (1978), Issues in Distributed Processing - An Overview of Two Workshops, Computer, Vol. 11, No. 1, pp. 22-26.

Elzas MS (1979), What is Needed for Robust Simulation in Methodology in Systems Modelling and Simulation, (ed.) B.P. Zeigler, et. al., North-Holland Pub., Amsterdam, pp. 57-91.

Enslow PH (1977), Multiprocessor Organization - A Survey, Computing Surveys, Vol. 9, No. 1, pp. 103-129.

Feldman JA (1979), High Level Programming for Distributed Computing, Comm. ACM, Vol. 22, Vol. 6, pp. 353-368.

Halin HJ, et al (1979), The ETH Multiprocessor Project: Parallel Simulation of Continuous Systems, Simulation, Vol. 35, No. 4, pp. 109-123.

Hoare CAR (1978), Communicating Sequential Processes, Comm. ACM, Vol. 21, Vol. 8, pp. 666-677.

Lamport L (1978), Time, Clocks, and the Ordering of Events in a Distributed System, Comm. ACM, Vol. 21, No. 7, pp. 558-565.

Liskov B (1979), Primitives for Distributed Computing, Proc. 7th Symp. on Operating Systems Principles, pp. 33-42.

Notkin DS (1980), An Experience with Parallelism in Ada, SIGPLAN Notices, Vol. 15, No. 11, pp. 9-15.

O'Grady EP (1979), Interprocessor Communication in Multiprocessos Simulation Systems, Proc. COMPCON, pp. 300-306.

Peacock JK, Manning E and Wong JW (1980), Synchronization of Distributed Simulation Using Broadcast Algorithms, Computer Networks, Vol. 4, No. 1, pp. 3-10.

Peacock JK, Wong JW and Manning EG (1979), Distributed Simulation Using a Network of Processors, Computer Networks, Vol. 3, No. 1, pp. 44-56.

Pritsker A and Young RE (1975), GASP-PL/I: A Pl/I Based Continuous/Discrete Simulation Language, Pritsher and Associates, Inc, Fayetteville, IN.

Reese RM and Sheppard SV (1983), A Software Development Environment for Simulation Programming, Proc. 1983 Winter Simulation Conference.

Reynolds PF (1982), A Shared Resource Algorithm for Distributed Simulation, Proc. 9th Annual Symp. on Computer Architecture, pp. 259-266.

Scherr AL (1978), Distributed Data Processing, IBM Systems Journal, Vol. 17, No. 4, pp. 324-343.