Architectural Considerations of the Parallel SIMULA Machine

M. P. Papazoglou*
University of Dundee Department of Electronics and Microprocessors, Dundee, UK

P. I. Georgiadis and D. G. Maritsas
University of Patras, Department of Computer Sciences, Patras, Greece

The development of problem-oriented hardware has become a possibility as a result of current technological advances. New machine architectures can now be defined, reflecting the high performance requirements which are set by specific application areas. Large-scale discrete time simulation is a problem area which demands high-speed processing, especially within the realm of real-time applications. The natural parallelism which characterizes many real-life dynamic systems and remains unexploited in the environment of uniprocessor machines, points decisively towards multiprocessor structures. Although the process-oriented simulation languages have proved very effective in accommodating this kind of parallelism in the corresponding simulation programs, their execution in the uniprocessor machines reverts the situation back to the unhappy reality of serial execution of otherwise independent tasks. SIMULA-67 is a process-oriented language specifically designed for advanced and complex software simulation products. The process structure of SIMULA allows for the definition of concurrent tasks within a SIMULA program which can be executed in parallel within the environment of a multiprocessor system. The analysis and the basic software architectural scheme for a parallel SIMULA machine (PSM) have already been reported by the authors in a previous paper. In the present paper a hardware organization of a PSM is presented. A system architecture is explored which is conceived so as to implement efficiently the execution algorithm (process co-ordination and synchronization algorithm) which has been outlined by the authors in previous papers. The architecture is based upon the master/slave system topology. It incorporates a central controller microprocessor and a number of satellite microprocessors. The interconnection circuitry between the microprocessor modules involves a time-sharing system bus and various programmable interrupt control units. Common and private memory modules reside in the system, and DMA transfers are employed to alleviate the controller's workload. The time operational features of the parallel SIMULA machine are also described.

INTRODUCTION

The development of problem-oriented hardware has become a possibility as a result of current technological advances. Much of the research in this area has focused around the development of high-level language architecture, the computer architecture that has been designed to facilitate the implementation of specific high-level programming languages. Apparently, new machine architectures can now be defined, reflecting the high performance requirements imposed by specific application areas.

A problem area that demands high-speed processing is large-scale discrete time simulation. SIMULA-67[1,2] is a general purpose programming language particularly offered for describing and efficiently simulating large-scale systems. The process structure of SIMULA-67 allows for the definition of concurrent activities decisively pointing towards parallel execution within a multiprocessing environment.

In this paper we analyse the basic architectural considerations of a parallel SIMULA machine (PSM) aiming to achieve fast processing rates.

The software architectural mechanisms for the PSM have already been covered in previous papers.[3,4] For clarity we briefly introduce here the main notions of this concept.

* Present address: University of Patras, Department of Computer Sciences, Patras, Greece.

The PSM uses a particular information structure containing a detailed record of SIMULA process activities. This information structure caters for the parallel evolution of SIMULA processes by providing suitable information obtained both at compile and run-time levels.

This information structure also denotes the identification of SIMULA processes on a producer (P)/consumer (C) basis. This distinction results from an individual syntax rule of the SIMULA language. The P/C process classification is fundamental to establishing parallelism in the PSM.

In order to ensure the correct evolution of SIMULA processes within the PSM an executive algorithm has been developed. This executive algorithm constitutes the kernel of the PSM operating system and has as main functions to:

(i) dispatch SIMULA processes to processors according to certain dispatching rules elaborated in Ref. 3
(ii) deal with synchronization phenomena either imposed exclusively by critical sections, or related to the synchronization imposed by specific SIMULA commands appearing within the critical sections. These two synchronizing modes have been exhaustively presented in Ref. 4.

In the following sections we propose the PSM architecture that is based upon the master/slave topology. This architecture incorporates a central microprocessor (mas-

CCC-0010-4620/84/0027-0254$03.00

ter) and a number of satellite microprocessors. The interconnection circuitry between the microprocessors involves a time-sharing system bus and various programmable interrupt control units. Common and private memory modules reside in the PSM, and DMA transfers are employed to alleviate the master's workload. Finally, the operational features of the PSM are also described.

PARALLELISM IN SIMULA

An attempt to achieve parallelism within the SIMULA environment must be oriented towards a successful coordination of control among the various processes comprising a SIMULA program. Parallelism between processes can be first investigated in the structure exhibited by the program text which can be thought of as comprising the program's static portrayal or its static level. A SIMULA program's static portrayal is composed of the program's definition structure including the various class declarations, the program statements and finally all the variables entitled to be accessed by any class declared within the simulation program. Such variables are referred to as 'system variables'. This means that parallelism detection can be initiated at program static level and prior to the actual execution of the SIMULA processes. This is due to the fact the program text reflects the structure of the computation.

The PSM environment should not only cater for the detection of disjointness but should also recognize the various interaction points contained within the class contexts of the program at declaration level. Process interaction is affected:

(1) Through the interruption of processes in the midst of their execution in order to issue certain SIMULA commands, possibly affecting their status or the status of some other process participating in the simulation program. Such commands are named 'communication commands' and include all the conventional SIMULA commands such as '(Re)activate', 'passivate', 'hold', 'cancel', 'wait' and the two primitive procedure calls 'detatch' and 'resume'.
(2) By the fact that a certain process is about to generate a new process via a call to procedure 'new'.
(3) Because the processes participating in the simulation program use system variables. System variables include all simple variables, procedures, arrays and references global to the environment of every SIMULA process participating in the parallely executed program.

The software tool that will detect potential parallelism at static level is a recognition mechanism. The SIMULA parallel process recognizer (SPPR) is a program that accepts SIMULA source programs as input, scans them on a line by line basis and generates tables containing all static level information, i.e. the system variables table or SV-table, and the class templates (CTs).

In particular, the SV-table shows the classes and therefore, the potential processes that refer to a specific system variable. On the other hand, the CTs provide the lifetime record of actions of a particular class, tabulating all communication commands issued on behalf of this class; they also show the chains of classes related on a producer/consumer basis.

As the program progresses through its various execution stages its behaviour starts becoming dynamic in the sense that processes start being generated and interacting asynchronously at unpredictable rates. It is a generally acceptable fact that the majority of SIMULA programs consist of a number of interacting rather than of disjoint processes.

Information that can be collected at static level is necessary but not sufficient in order to establish parallelism at execution level. Indeed, precedence relations dynamically evolving among the various processes, as well as the finite number of processors in a multiprocessor system, establish requirements imposing run-time action. Precedence relations accrue from the event-time attribute inherent in each SIMULA process. It is exactly the combination of information collected both at compilation and run-time levels that enables the parallel executable processes (or groups of process statements) to be efficiently determined at any time.

To accommodate run-time information the two-way circular list (SQS) of the uniprocessor SIMULA system is appropriately extended. This extended SQS (ESQS) along with the SPPR output tables compose the 'SIMULA process interaction structure' (SPIS) (Fig. 1). In addition to the SPIS there must be introduced a mechanism for efficient interprocess communication, so that the various SIMULA processes constituting a simulation model can transmit requests and wait for completions of results. There must also be a mechanism acting as a processor manager to start up any processor in the system and interrupt it when requested, and an additional mechanism to synchronize and share processors' time among the SIMULA processes. All these responsibilities in the PSM are absorbed by a fundamental system software module called the 'executive algorithm'. In reality the executive constitutes the kernel of the PSM operating system and is responsible for the process flow management in the PSM.

PROCESS FLOW MANAGEMENT

The executive algorithm is needed to guarantee the correct flow of SIMULA processes in their parallel environment. The main function of the executive algorithm is to control the evolution of the system in terms of co-operation of processors with processes so that determinacy is not violated.

The executive algorithm provides mechanisms for:

(a) interprocess communication, so that the various processes can transmit requests (commands) and wait for completions
(b) management of processors by starting up the various processors in the multiprocessor system and by handling their interrupts
(c) sharing CPUs' time among the processes by allocating–deallocating processors to them when appropriate.

The executive algorithm incorporates specific procedures which overcome issues arising during parallel operation of processors, synchronization problems and deadlock embracements. In the following we discuss briefly the three procedures embedded in the executive algorithm, i.e. the initializer, the dispatcher and the synchronizer.

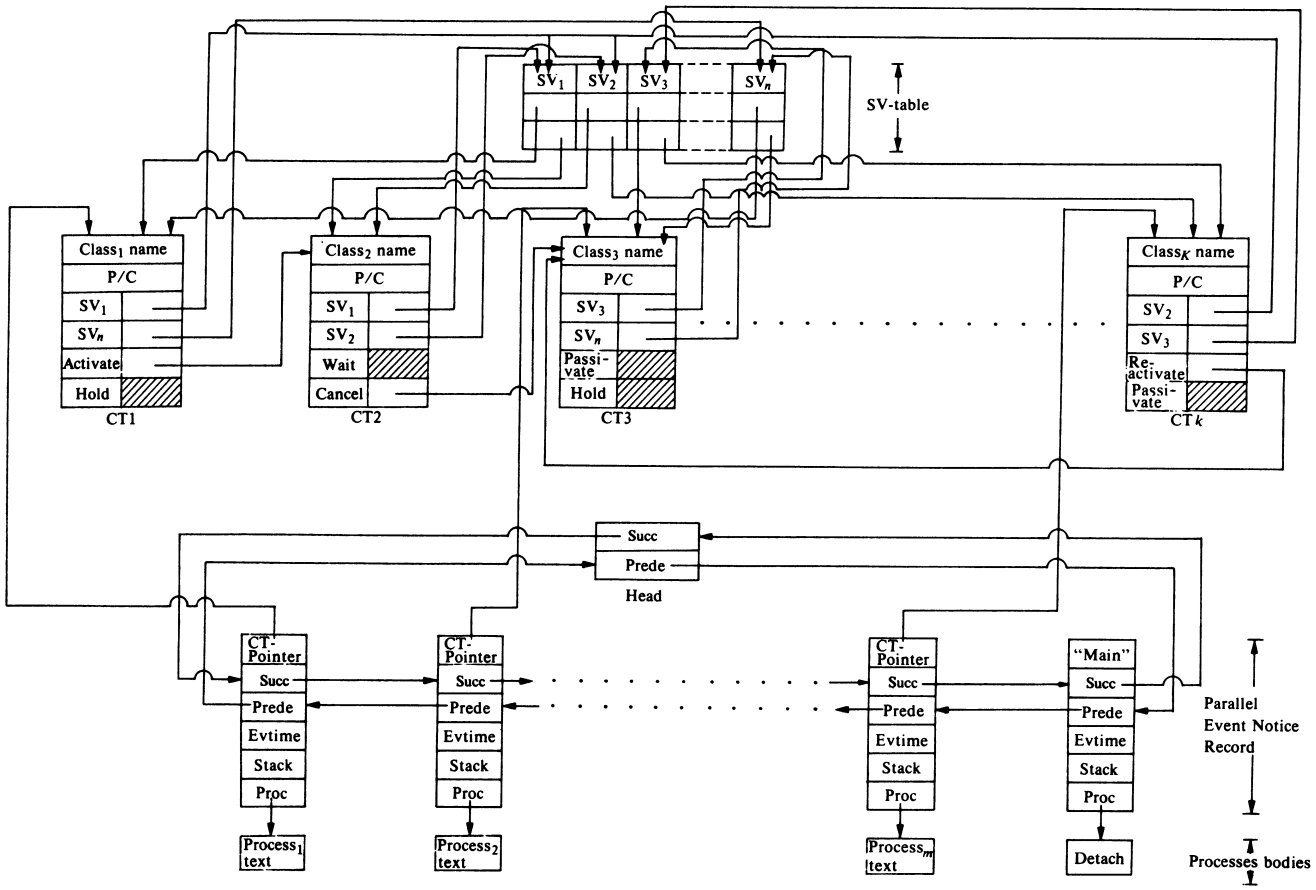

Figure 1. SIMULA process interaction structure (SPIS).

The SIMULA uniprocessor system orders the various processes within the SQS, during the initial phase of SIMULA programs, according to the user program statements so that the program execution can commence. The same approach is retained within the PSM; the only difference is that the process ordering is now taking place in and affects the ESQS, while the initializer undertakes the task of allocating processes to processors.

After the initial dispatching of processes among the various system processors, and as the simulated system progresses, its behaviour becomes dynamic, as frequent activation, removal and suspension of processes occur. As a result, a continuous rearrangement of processes inside the ESQS is effected; thus more sophisticated dispatching rules are now required. The application of these dispatching rules has two main objectives:

1. To maximize the number of processors that are used at any given time.
2. To minimize the system reconfiguration overheads. It is the responsibility of the dispatcher to apply the dispatching rules, described in Ref. 3, and fulfil the above objectives. Finally, the synchronizer caters for implementing proper sharing relationships on system variables accessed by these SIMULA processes. This approach should be implemented in a well-defined manner so as to allow deterministic behaviour on behalf of the program and preserve integrity of the system variables, as well as to design this synchronizer mechanism efficiently. Thus time-dependent interaction points among processes to be executed in parallel must be gathered from the SPIS.

All of the above mentioned principles appear in the hardware and functional analysis of the PSM which will be covered in the following sections.

THE PSM HARDWARE STRUCTURE

The PSM comprises a number of microprocessors connected on a master/slave topology basis. The slave (satellite) microprocessors communicate asynchronously with the master processor by means of interrupt signals resolved by the PSM interrupt circuitry. For the PSM purposes the satellite CPUs contain their own private memory modules, while using the system common memory. A communication path is also provided to guarantee efficient interfacing between the master and the satellite processors. This interface circuitry consists of the various PIA devices attached to the master and satellite processors as shown in Fig. 2. In essence, each individual PIA provides all necessary communication and hand-shaking operations. The interface circuitry also comprises the system interrupt controllers and the various DMA devices.

The PSM is an interrupt driven system. Therefore there exists a need for the resolution of multiple interrupts. The PSM interrupt circuitry contains a single PICU

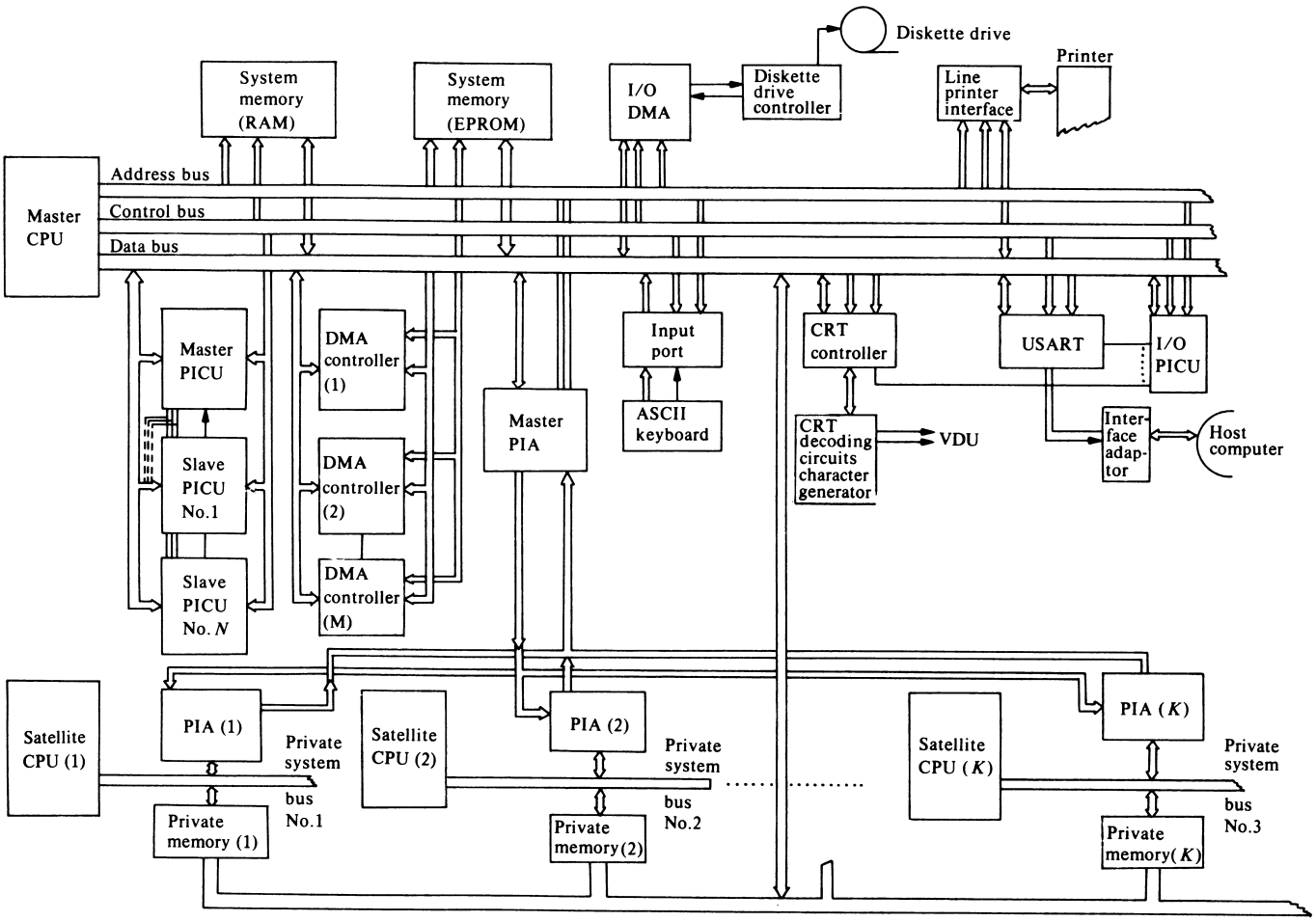

Figure 2. The PSM hardware structure.

(programmable interrupt control unit) termed the master PICU which controls a specific number of slave PICUs in cascade. The DMA control circuitry relieves the master CPU of the burden of large data transfers towards the satellite circuitries.

The satellite circuitries comprise the satellite processors, their private PIAs, their private buses and private RAM memory modules.

The private memories are dual port memory devices and obviously communicate with their associated processor by means of their private bus, although they can use the system bus when necessary.

It is apparent that there exists a continuous communication path between the various memory modules and the system shared memory. On the other hand, private memory modules are prohibited from exchanging messages with each other, since all memory intercommunications are accomplished via the PSM shared memory. The system memory (shared memory) consists of RAM and EPROM modules. The RAM modules contain the SIMULA executable code and the information structures.

The PSM uses a time-shared bus topology. Although, time-shared bus configurations suffer from throughput limitations the design objective of the PSM was such that all potential bottlenecks were diminished.

The outline of the PSM functions does not take into account the i/o facilities of the PSM which appear in Fig. 2. This is due to the fact that the i/o techniques followed are standard and have been extensively covered in the relevant literature.

THE PSM FUNCTIONALITY

The PSM is a tightly coupled multi-microprocessor system[5] and accepts SIMULA executable code as compiled by a supporting host mainframe device. The host machine system could be any general purpose computing system supporting the SIMULA compiler, or even the S-PORT system[6] which aims at SIMULA portability, thus simplifying the choice of the host machine. Along with the SIMULA code the PSM accepts the appropriate information structures.

These structures are constructed by a software preprocessor module (SPPR) resident on the host machine, scanning source SIMULA programs only.

Figure 2 depicts the PSM architecture which could be classified as a master/slave multiprocessor configuration. The PSM can communicate with the external world (host computer) by means of a USART module. This device accepts the code from the host machine in a serial fashion and converts it into parallel format for the PSM requirements. The PSM shared memory (RAM) receives the SIMULA executable code, and the information supporting software modules. The system memory also

consists of the EPROM memory modules containing the executive algorithms. These algorithms support and guide the various PSM functions.

SIMULA processes that can be executed in parallel must be distributed to the private memory modules by means of DMA techniques. It is evident then that the process executable codes are loaded within the private RAM modules with no processor involvement, since the master μCPU is electrically isolated from the system bus during DMA cycles. It must be mentioned that process executable codes are placed within the private circuitry RAMs according to their priorities. Private satellite modules that have already been loaded with process executable codes can commence execution while the code transfer technique is in progress.

In the case that the transfer tasks of the process executable codes into the private satellite modules has not terminated, and a particular process already executing reaches a point where there arises a necessity for a synchronization or co-operation activity, then this process must be suspended. The process must remain suspended until the transfer task of the process executable codes has been terminated. The flow chart of Fig. 3 shows clearly the details of the transfer-code task.

SIMULA processes during their parallel evolution are normally highly interactive. This interaction results in process co-operation and synchronization activities. Process co-operation activities are the outcome of the execution of certain SIMULA commands called communication commands,[3] e.g. hold (t), passivate, reactivate, etc. Process synchronization activities are due to the sharing relationships imposed on common variables accessed by SIMULA processes. We distinguish two synchronization modes:[4]

(i) the V-mode specifically related to the synchronization imposed by the critical sections within the various processes
(ii) the C-mode related to the synchronization imposed by the SIMULA communication commands within the critical sections.

Whenever a satellite processor attempts to execute a co-operation or synchronization activity it issues an interrupt signal. This interrupt signal is finally received by the master PICU which forces the system to suspend its executional progress. In the case that several interrupts occur concurrently the master PICU must determine which request must be serviced first, based upon the priority of the events causing the interrupts. The priority of the events causing the interrupts is determined from the event time attributes of each executable SIMULA process.[1] The interrupt mechanism is implemented in such a way as to preserve the volatile environment of the satellite processors and to direct the master CPU to resume execution of the appropriate interrupt service routine. The PSM employs a non-pre-emptive policy[7] in the case of simultaneous interrupt requests.

Dispatching of processes to satellite processors is achieved by a procedure of the executive algorithm running on the master CPU and is based on a pre-emptive strategy. The dispatching function is affected by the execution of certain co-operation activities, such as the execution of some specific communication commands that take as parameters references to processes. A set of dispatching rules have been formulated in Ref. 3 in order

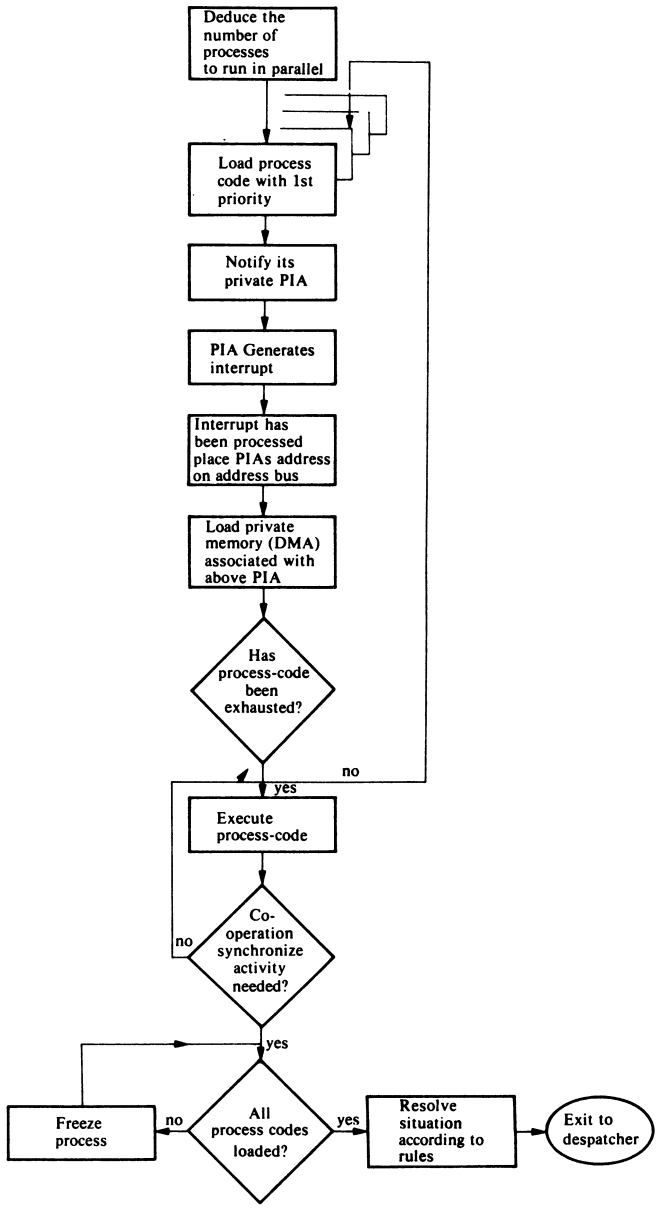

Figure 3. The process code transfer function.

to implement the proper process distribution within the PSM.

Hence the PSM resolves all problems emerging during the parallel evolution of SIMULA processes in a strictly deterministic fashion, so as to ensure program consistency and data integrity while achieving fast processing rates.

SYSTEM SIMULATION

To study the system process load variations and its overall behaviour under the direction of the executive, the PSM has been simulated by using a slightly modified version of the OASIS software prototyping system.[8,9] OASIS is an extension of the SIMULA programming language which supports the development of prototype software for multiprocessor architectural configurations. OASIS

has been designed for developing prototypes of distributed software and simulating the execution of these software prototypes on modelled hardware. The OASIS implementation provides both a specification of network software and a prototype which can be simulated to verify design adequacy and predict network performance. It defines abstract types in the form of prefixed SIMULA classes for characterizing the device, memory and processor hardware of a given multiprocessor system for implementing distributed tasks, and for simulating the execution of these tasks on the modelled hardware. These abstract types are DEVICE, PROCESSOR, MEMORY, SEGMENT and MODULE; they respectively represent arbitrary computer system 'peripheral devices', 'CPUs', 'information storage devices', 'units of information' and 'chunks of executable code'. Moreover, each of these five abstract types defines attributes in the form of SIMULA variables or procedures representing state information or actions performed by objects of that type.

The PSM hardware and software sections have been implemented either as instances or as instances of extensions of the above abstract types; thus specific features of the PSM design have been efficiently described. Hence the PSM computer model structure has been accurately described in terms of both software and hardware, and some useful conclusions have been reached. The structure of the PSM simulator along with the results obtained and the deductions drawn will be reported analytically in a future publication.

CONCLUSIONS AND FUTURE DIRECTIONS

The main issues involved in both the hardware and software design of a multiprocessor system for parallel execution of SIMULA programs have been presented.

An efficient hardware structure has been shown to execute SIMULA programs in parallel under the directions of a versatile system executive whose design was strongly influenced by the language features.

This research project is now in progress and certain simulation programs have been developed in both the directions of the hardware and software areas of the PSM. This simulation study intends to draw deductions concerning the statistical analysis of evolution of SIMULA processes within the PSM structure and also estimate the percentage amount of involvement of each satellite CPU in the various computations of SIMULA programs. This simulation project will, eventually, draw up guidelines concerning the performance of the PSM structure. It will also finally determine the optimal number of satellite processors in conjunction with a wide range of sample SIMULA programs related to the field of large-scale advanced simulations.

REFERENCES

1. O. Dahl, B. Myhrhaug and K. Nygaard, Common base language. *Publ. No. S-22*, Norwegian Computing Centre, Oslo, Norway, October (1970).
2. W. R. Franta, *The Process View of Simulation*, Elsevier, North Holland Publishing Co. (1977).
3. P. G. Georgiadis, M. P. Papazoglou and D. G. Maritsas, Towards a parallel SIMULA machine. *8th Ann. Symposium on Computer Architecture*, 263–278 (May 1981).
4. M. P. Papazoglou, P. G. Georgiadis and D. G. Maritsas, Process synchronization in the parallel SIMULA machine. *Proceedings Intern. Conference on Parallel Processing*, 297–299 (August 1981).
5. B. A. Bowen and R. J. Buhr, *The Logical Design of Multimicroprocessor Systems*, Prentice-Hall, Englewood Cliffs (1980).
6. P. Jensen, S. Krogdahl, O. Myhre, P. S. Robertson and G. Syrrist, Definition of S-code. *Report No. 672*, Norwegian Computing Centre, December (1980).
7. G. D. Kraft and W. N. Toy, *Mini/Microcomputer Hardware Design*, Prentice-Hall, Englewood Cliffs (1979).
8. B. W. Unger, D. Bidulock and J. Parker, OASIS reference manual. *Report No. 81/58/10*, Computer Science Dept., University of Calgary, March (1981).
9. P. Belanger, D. Bidulock, C. Hankins, N. Jain and B. Unger, An OASIS simulation of the ZNET microcomputer network. *Report No. 81/67/19*, Computer Science Dept., University of Calgary, June (1981).

Received June 1983