DESIGN OPTIMIZATION FOR A SPECIAL-PURPOSE MULTIPLE-COMPUTER SYSTEM

by

C. F. Summer, Naval Training Equipment Center, Orlando, Fla.

and

R. O. Pettus, R. D. Bonnell, M. N. Huhns, and L. M. Stephens
University of South Carolina, Columbia, SC

ABSTRACT

The design and performance analysis of the architecture of a special-purpose multiprocessor is presented. The architecture is a hierarchically structured and functionally distributed type. Its operating system is a multilevel structure implemented in an optimal combination of hardware, firmware, and software. This architecture is suited to any application, such as process control or system simulation, in which the basic computational tasks do not change in time.

Each processor has a dedicated memory space in which program tasks are stored. In addition, there is a system bus to a global memory which is used primarily for communication among the processors. To minimize contention for this system bus, selected areas of global memory are duplicated at each processor. This allows the processor to obtain needed information by using a local bus rather than the global, system bus. All write operations to the shared memory are global and the information is duplicated at processors having that address. Read operations then become primarily local and can occur in parallel.

Control functions are distributed among the processors; the scheduling and execution of control and application tasks are governed at each processor level by a local, real-time operating system. This local operating system is implemented primarily in firmware to minimize overhead. However, the control structure is designed to be independent of implementation so that a variety of processors can be utilized together. Moreover, it is possible to add to each local processor an additional subprocessor which implements the operating system.

I. INTRODUCTION

Much of the previous work in multiprocessor systems has addressed the problems of allocating system resources in a general-purpose computing environment. In these systems the computational requirements change as a function of time as various tasks use different resources in a random manner. This paper discusses the application of multiprocessor capabilities to a fundamentally different problem in which the computational load is essentially independent of time. Such applications occur in process control and in simulations of physical systems. The computations in these applications are repetitive in that a certain number of calculations are required to move the system from one finite state to the next. Computations for one state must be completed before calculations for the next state may begin. This requirement imposes a structure on the computational load because it is known in advance which computational tasks must be completed before the system moves to the next state.

A multiprocessor is utilized in order to exploit the capabilities offered by parallel processing. A single processor computer system may not complete computations quickly enough to provide real-time responses, especially when a large number of features are included in the system model. Partitioning the program into parallel processible units and using a multiprocessor system is one method of obtaining the required computational speeds. However, once a program has been partitioned, a problem of communication among the processors is introduced, a problem compounded by the highly-coupled structure of many system models . If the multiprocessor system is to be effective, communication of data from one processor to another must be optimized so that needed data can be passed with minimum delay to other processors. Techniques for communications optimization are discussed in detail in section III.

One way to ameliorate the communications problem is to minimize the transmission of control information. This can be accomplished by distributing portions of the control function of the system to individual processors. These operate autonomously from the system control processor as discussed in section IV. Performance analysis, section V, centers around the concept of the speed-up factor, a ratio of the computation time required in a single processor to that in a multiprocessor system. Results and conclusions are found in section VI.

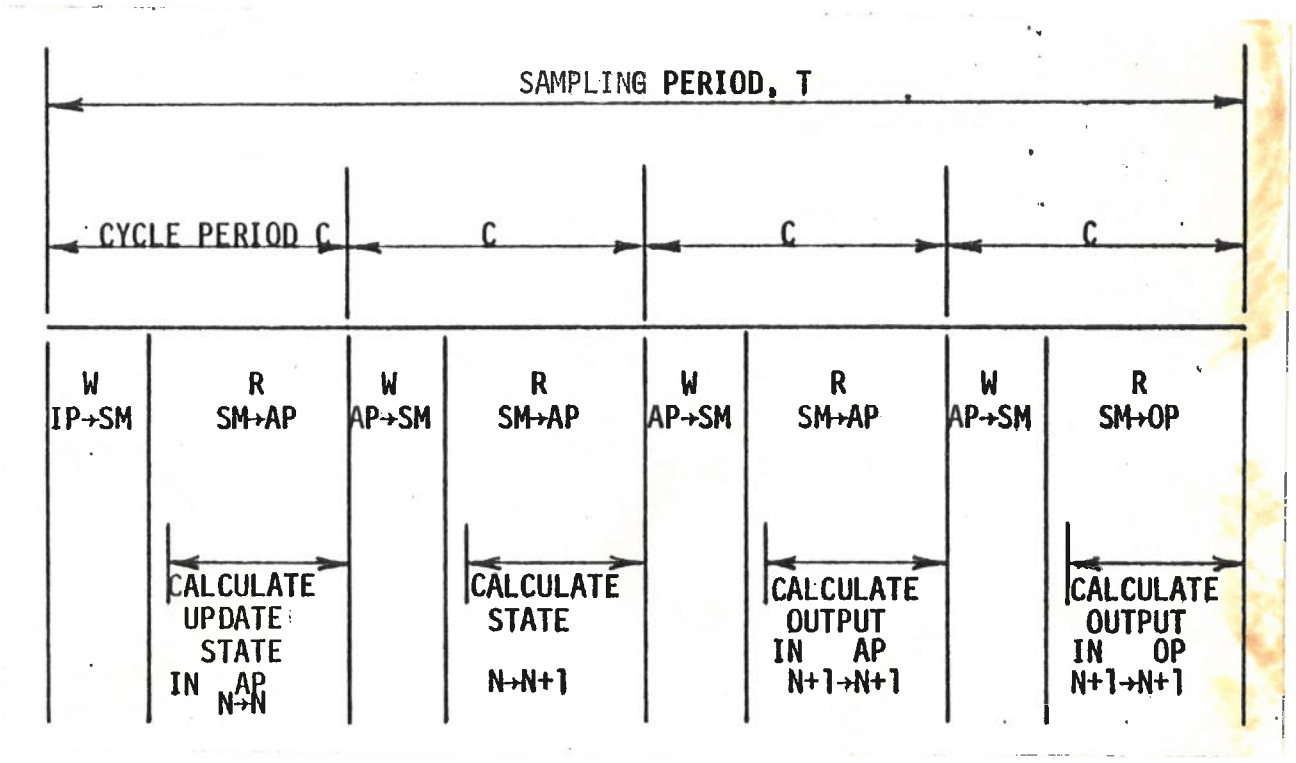
The architecture selected for the multiprocessor system described herein is straightforward. All processors are fundamentally identical although one is given overall system control responsibility by virtue of its priority. Each processor has a dedicated local memory for program and operating system storage. Communications among processors are handled via a shared memory. The details of this architecture are further elaborated in section II.

II. ARCHITECTURAL FEATURES

The system considered here is designed to simulate a physical process such as the flight characteristics of a complex aircraft. This process has a large but finite number of state variables which must be updated periodically. The overall program is fixed once the system model is determined; only the state variables change in time. Because the fundamental process being modeled does not change, the program which implements the model also does not change.

In a flight simulator, for example, the state variables are updated at a rate determined by the dynamics and outputs of the system. If at $t=T_0$ a trainee pulls back on a simulator control yoke, then at $t=T_0+\Delta t$, one sampling time later, the simulator cockpit must be moved accordingly. The computations which carry the simulator from one state to the next can be decomposed as a four-fold process as depicted in Figure 1 and described below.

First, the sampled input parameter (control yoke position) is mapped into a change in elevator position. This change is called the update state calculation



AP	=	application processor
SM	=	shared memory
OP	=	output processor
IP	=	input processor

Figure 1. Data transfers for a sampling period with four cycles.

and must be communicated to all those processors making calculations which depend on it. Secondly, the next state is calculated based on the present value of the state parameters, such as velocity and wind direction, and any changes in input parameters, such as elevator position.

The third step in the process consists of computing output quantities such as changes in cockpit position and instrument readings. In the fourth phase, the output values are transmitted to the simulator mechanism and, in this example, the cockpit moves in response to the change in control yoke position.

The sequence described above makes certain demands on the interconnection structure of a multiple-computer system. Star and loop methods have been considered but discarded for the following reasons:

1. In a star configuration each processor is connected to every other processor. If there are n processors, there are n-1 bidirectional communication links to a given processor. It is difficult to control communications in such a structure because the cooperation of the processors being addressed is required and there are numerous paths in the network. If the addressed processor is busy with computations, its execution would have to be suspended during the communication.
2. In a loop configuration, communications pass through each processor, a path which introduces buffer and repeater delays in an already time-critical system.

The common bus structure shown in Figure 2 has been selected as being most desirable for the application described here because this structure provides a means of orderly communications between processors. The next question addressed is that of determining the better communications strategy: message-based or shared-memory type. The message-based method has been discarded in favor of a shared-memory system because the shared-memory method allows all

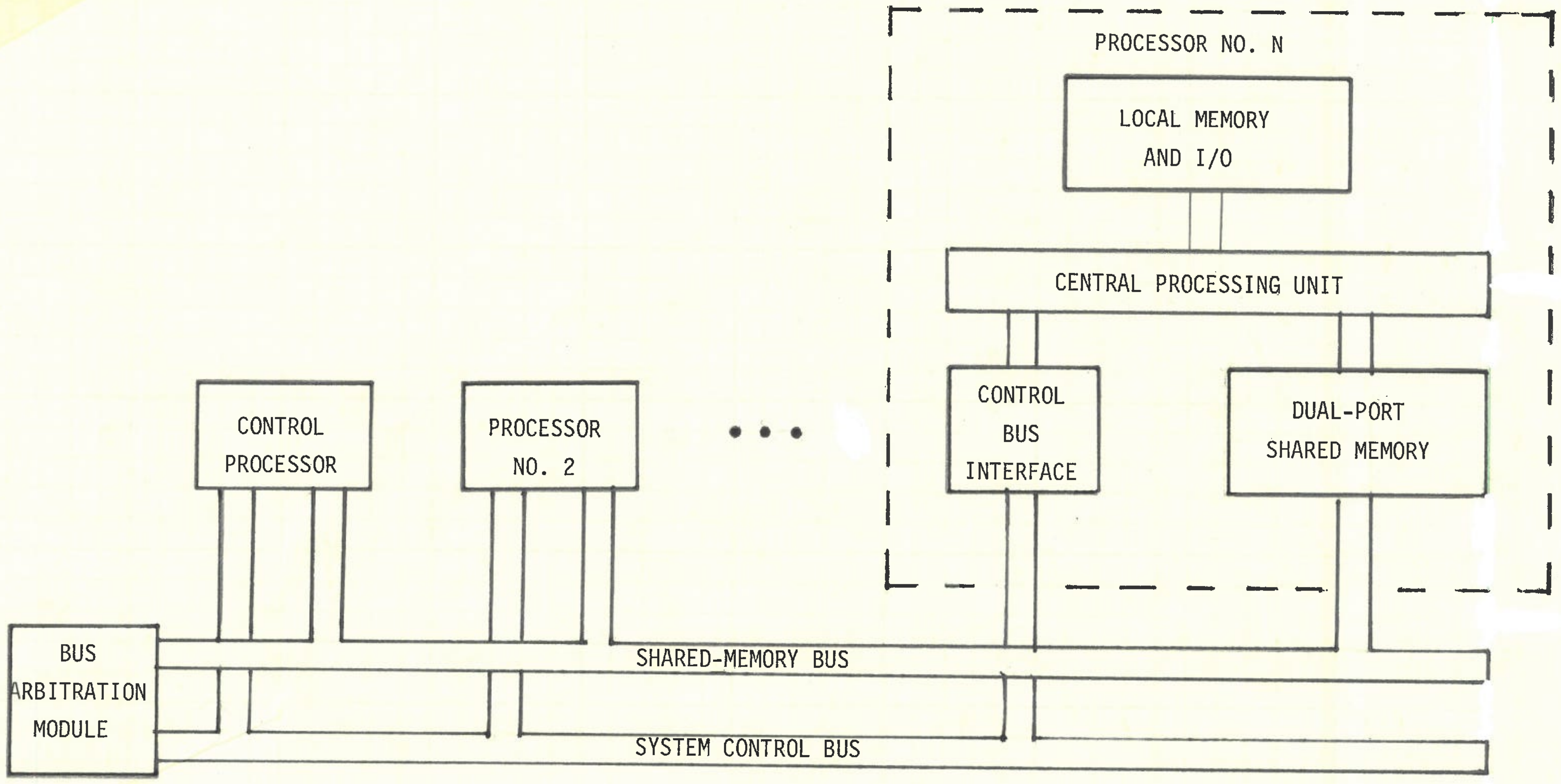


Figure 2. The global bus structure for the multiple-computer system.

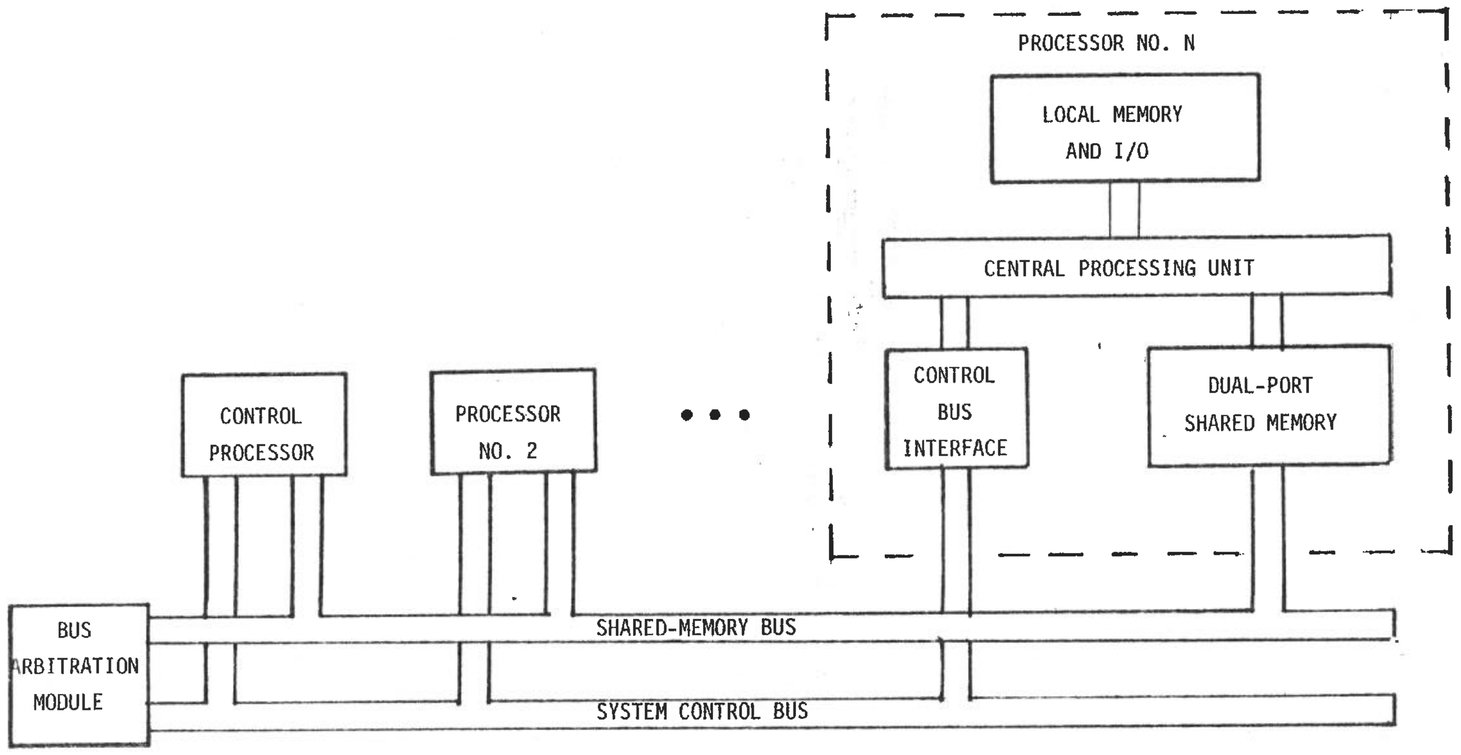


Figure 2. The global bus structure for the multiple-computer system.

communications to occur without the cooperation of the processors involved. A processor may send its newly calculated parameters to shared memory whenever the bus is available. It need not wait until the recipient processor is ready to receive that data. This can be a significant advantage if, for example, the calculations of one processor are required by several others. The same advantages occur when a processor requires information from another processor.

Coordinating the processors and insuring that computations are not started until all updated data is available is the task of the real-time operating system. Its operation is described in section IV.

III. MINIMIZING BUS CONTENTION

The key to successful operation of a multiple-instruction-stream, multiple-data-stream (MIMD) computer is effective communications among the processors. As discussed in the previous section and shown in Figure 2, there are two system buses--one for communicating data and the other for communicating control information--which are common to all of the processors. The most critical system resources are these global buses which, by being shared by all of the processors, become the limiting factor in the overall performance of this multiple-computer system. It is thus crucial that the design and utilization of these buses be optimized.

The architecture of the entire system can be designed to minimize bus usage. Most of the system control functions are distributed among the processors by providing each with a local real-time operating system. Also, because the programs to be executed are fixed, each processor is assigned its function in advance. Hence, although one processor is designated as a control processor, it needs to communicate only a minimum of control information during normal system operation. This control information is transmitted on the control bus so as not to interrupt the data flow on the other bus.

One way for processors to communicate is by writing messages and results into a shared memory where other processors can access this information. For the MIMD system described herein, all of the system memory is distributed among the processors. Part of the memory for each processor is local and can be accessed only by that processor. This allows most run-time memory operations to be local, thereby avoiding contention for the global buses. The rest of a processor's memory is global and available to all processors for memory-write operations. This global portion is designed in a dual-port configuration so that it can be read locally while being written globally. Also, all processors can read in parallel without any possibilities for contention or deadlock. By removing all global read operations from the bus, the bus traffic is reduced by much more than half, an important reduction for this vital system resource.

As an example of this reduction, if a parameter calculated by one processor is needed by four other processors, a simple shared memory would handle this transfer in five cycles (one to write and four to read). With the shared memory duplicated at each processor, only one cycle is required to simultaneously write the parameter to all processors which need it. The destinations for a parameter are determined by its location in the memory address space. The read operations then occur locally and independently.

An additional architectural feature which maximizes the bandwidth of the global data bus is synchronous operation. This reduces the overhead associated with each data transfer and allows most data transfers to be scheduled.

The utilization of the bus can be further minimized because the system is to be used for a single application. The program for this application can be partitioned into tasks and assigned to processors for execution in a way that minimizes the interprocessor communications. Also, the communications can be scheduled in advance to minimize idle period for the bus and wait periods for processors, both of which add to communications overhead. Neither of these optimizations are readily available in a general-purpose MIMD system.

IV. REAL-TIME OPERATING SYSTEM

The operating system is distributed among all of the processors. Each local operating system has two major functions: it implements a virtual machine structure and handles the chores normally associated with a real-time task manager. In addition,it isolates a programmer from the details involved in the passing of parameters between tasks.

A. The Virtual Machine

An individual processor may communicate with the rest of the system via the system buses, or with an external device via the I/O interface bus. The external communications are controlled by the programs executing in each processor but all interprocessor communications are handled by a virtual machine implemented by the local operating system. Use of the virtual machine removes much of the system dependence on any hardware characteristics of the individual processors. The role of the virtual machine is shown in Figure 3. The system structure is symmetrical in that the interface between the control program and the individual processors is the same as the interface between these processors and external devices. In addition,the interface between the control processor and the external world is handled by the virtual machine in the same manner as for the interface between the other processors and the system bus. This structure increases the extensibility of the system, allowing more than one system of multiple processors to be linked together.

A state diagram of the virtual machine is shown in Figure 4. The machine has five states: HALT, WAIT, COMMUNICATION, EXECUTIVE, and USER. The HALT state is used to take a processor off-line for an indefinite length of time. The

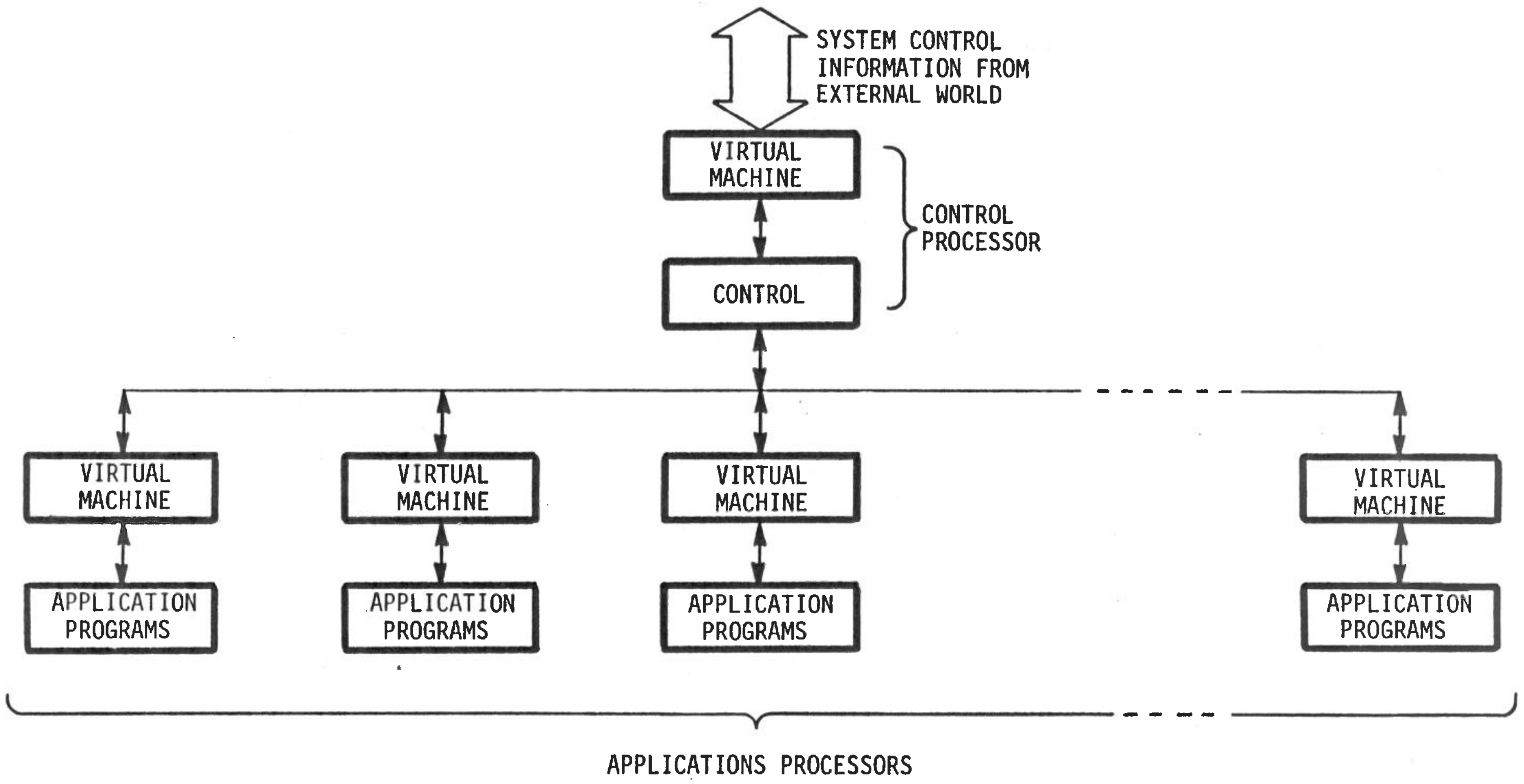
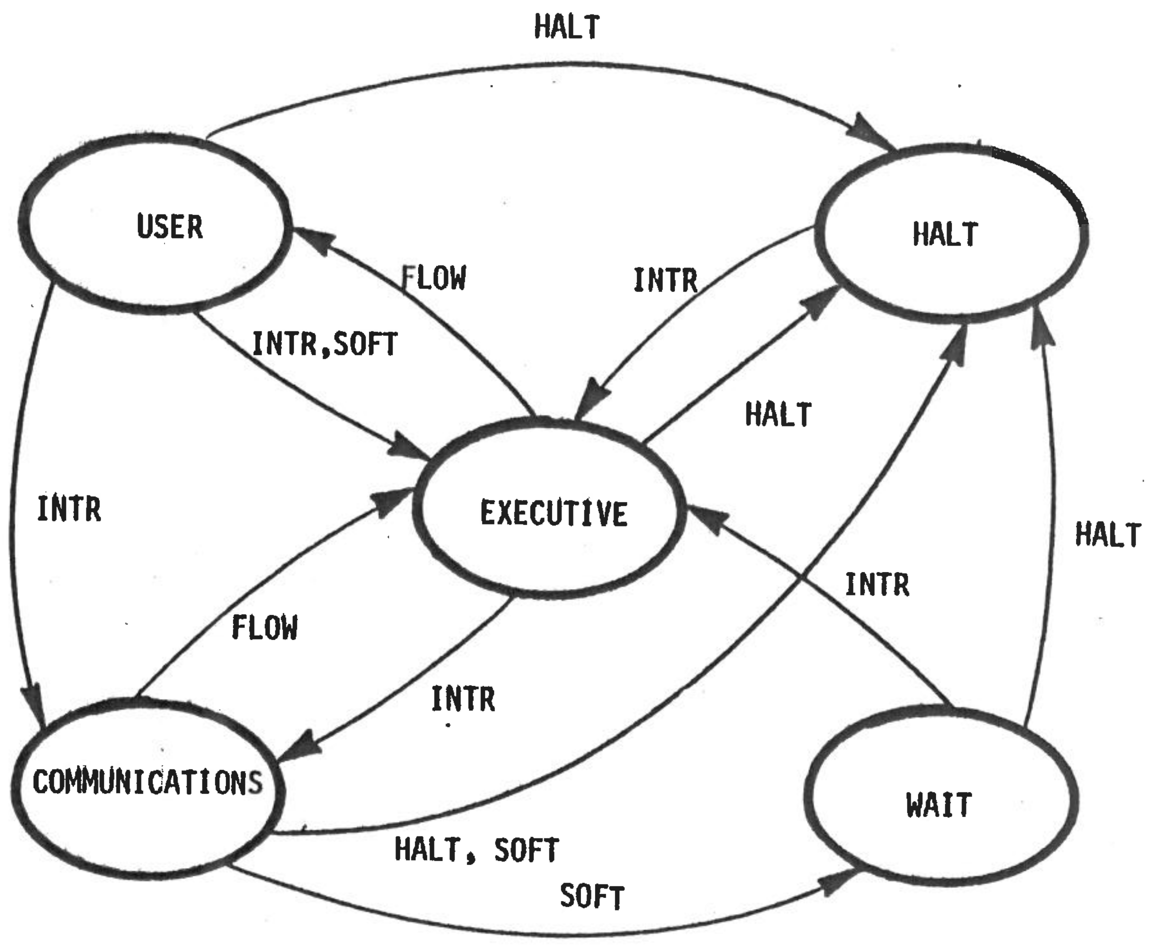


Figure 3. Operating system virtual machine structure.



INTR:	INTERRUPT
HALT:	HARDWARE HALT COMMAND
SOFT:	SOFTWARE CONTROL
FLOW:	NORMAL PROGRAM FLOW

Figure 4. Virtual machine state diagram.

WAIT state is similar to the HALT state but is used for synchronization. A common control line can cause all waiting units to enter the EXECUTIVE state simultaneously. Most of the normal operating system activities, such as the scheduling of tasks, take place in the EXECUTIVE state. The USER state is used to run the actual programs.

B. Run-Time Structure

A simplified flowchart of the operating system is shown in Figure 5. Entry to this system is by a software or hardware interrupt. The interrupt handler has a structure which is similar to that of the CASE statement. This allows the operating system to have a one-in, one-out structure even though there are multiple interrupts. Normally control is passed to the scheduler; however, under some conditions an exceptional task may be activated. Exceptional tasks include the following:

(1) The supervisor-call handler

(2) The error handler

(3) An initialization procedure

(4) The communications, halt and wait states of the virtual machine.

A flowchart for the handler of the exceptional events is shown in Figure 6.

The supervisor-call handler is a mechanism which provides system services to the executing programs. The supervisor calls allow scheduling of tasks, time management, and intertask communications control. Semaphore and message buffers are available as supervisor calls and are the main techniques used for inter-task communications and control. In addition, flags are used to implement a conditional critical region. Executing programs issue supervisor calls by using software interrupts.

The occurrence of an error, such as an attempt to address nonexistent memory or to divide by zero, causes a trap which activates the error handler. The error

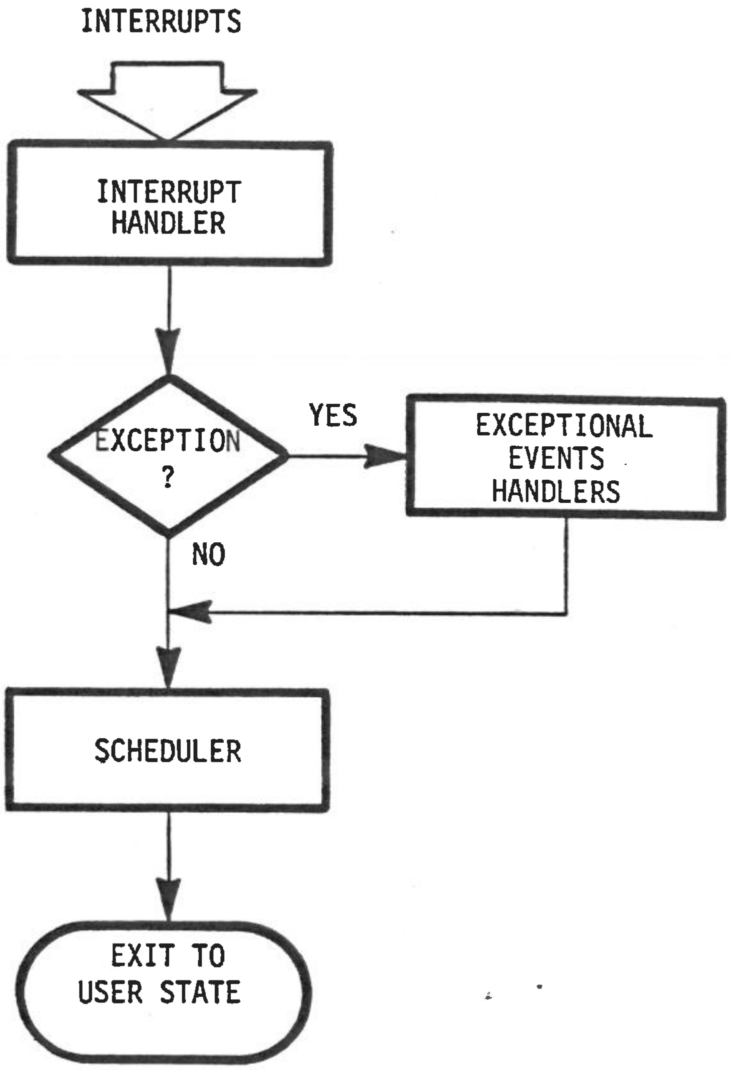


Figure 5. Simplified flowchart of operating system.

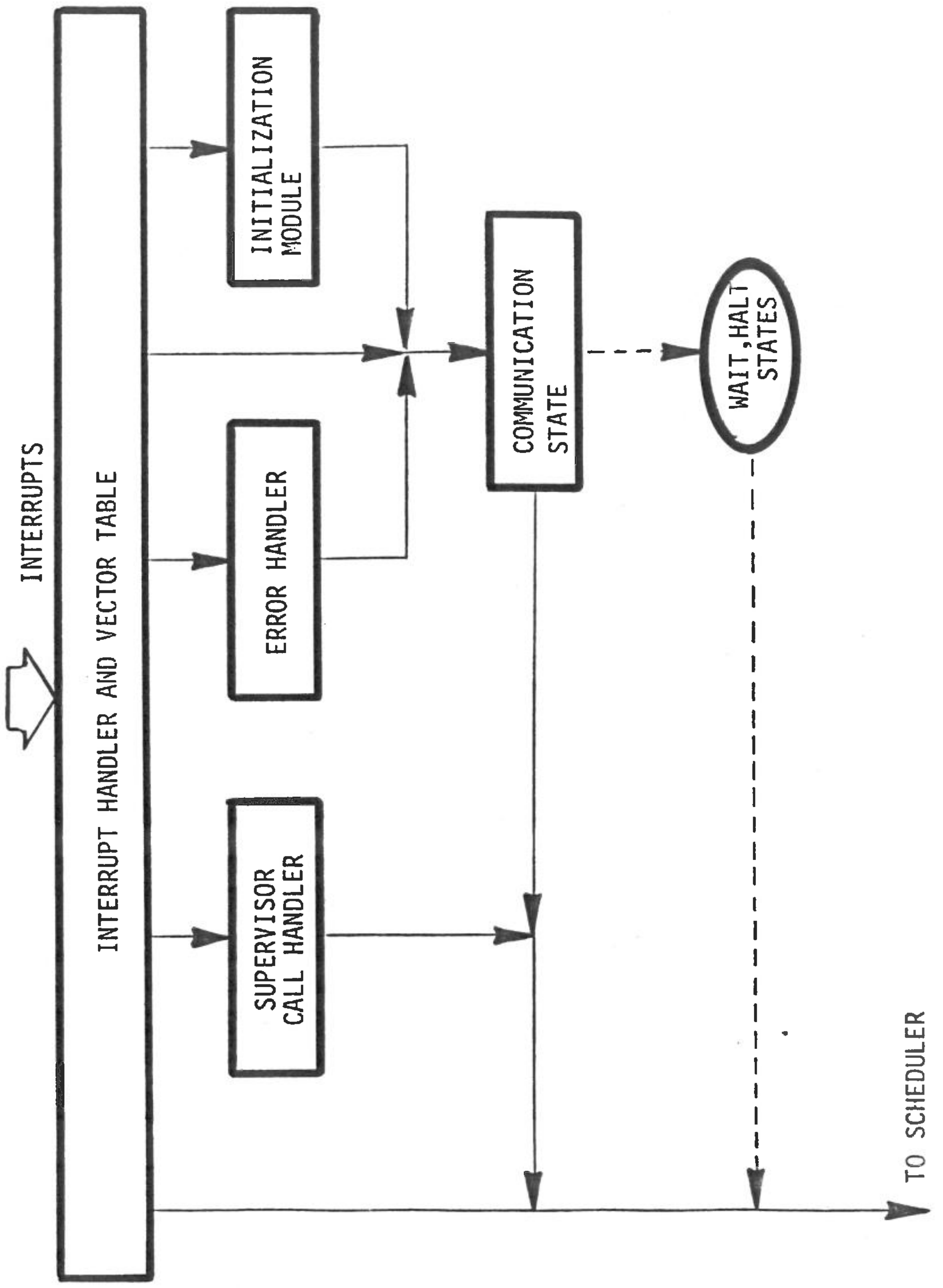


Figure 6. Exception handlers.

handler prepares a message, with information such as the machine state and identity of the active task, and passes control to the communications state where the message is then sent to the control processor. The task may or may not be restarted, depending upon the severity of the error.

The communications state is used for the transmission of programs, data, and control information between the control processor and other system processors. All messages sent to a processor in the communications state are interpreted by the virtual machine. The control processor does not have to be involved with any hardware details of the processor with which it is communicating.

The scheduler is shown in Figure 7. It uses single-level dynamic priority assignment and pre-emptive scheduling with resumption. The three main components of the scheduler are the event queue handler, the system program handler, and the application task scheduler. The event queue is the mechanism used to handle all events scheduled to occur either at some specific time or after an elapsed time. When an event timer interrupt occurs, the event queue handler is flagged to run by the interrupt handler. When the scheduler is entered,the task at the top of the event queue is activated. This insures only that the task can compete for processor time, not that it will run.

System programs implement operating system functions that are matched to a particular application. These programs are executed until completed whenever they are scheduled.

The application task scheduler compares the priority of the currently active task, if any, with the highest priority task in the queue. The higher priority task is scheduled and control is passed to that task. If there are no currently active tasks a diagnostic program is run.

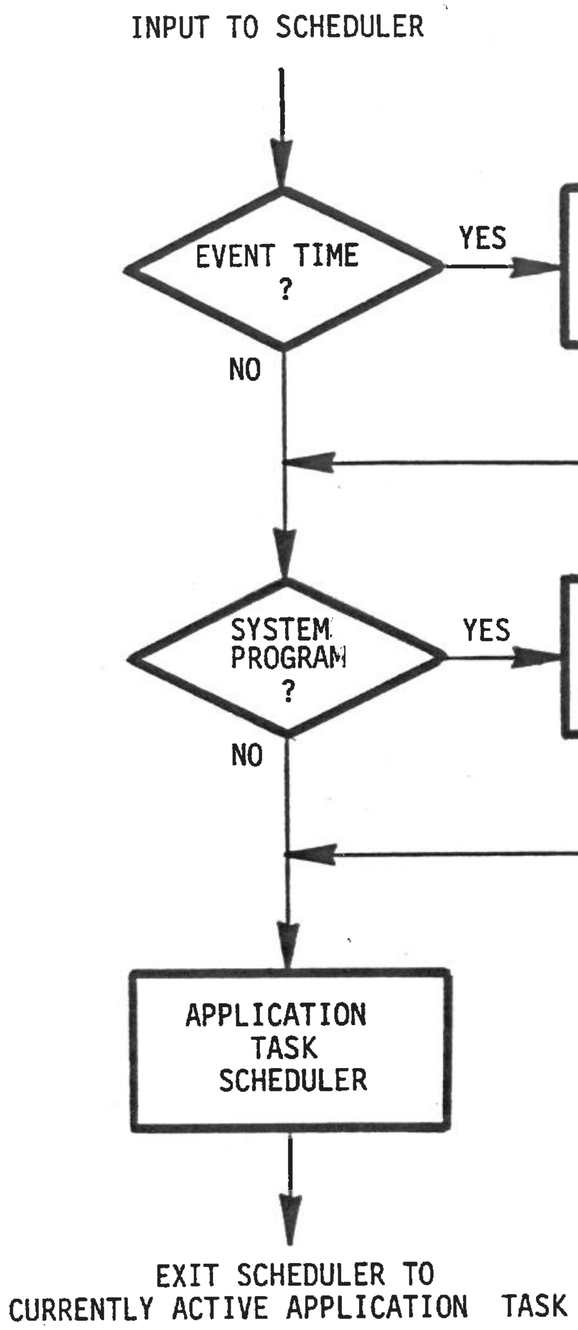


Figure 7. Flowchart of scheduler portion of operating system.

C. Implementation of the Local Operating System

The local operating system was designed to work on a wide variety of processor types. During the design phase, coding was performed in PASCAL. However, the first actual test version was coded in assembly language for the Motorola 6809 microprocessor. All of the basic operating structures were designed to allow flexibility in the coding of the machine-dependent portions of the code. The PASCAL version defined all the standard features and became an important part of the documentation.

V. PERFORMANCE ANALYSIS

As has been noted by Kober[1], how well the processing power of a multiple-computer system can be utilized (i.e., its efficiency) is a function of three major factors:

1. The organization and architecture of the system.
2. The number and power of the individual processors.
3. The type of application program.

One measure of the efficiency of a multiple-computer system is the speed-up ratio, β, defined as:

$$\beta = \frac{T_S}{T_P}$$

where T_S = the execution time needed for the sequential computation of the application program.

T_P = the execution time needed for the parallel computation of the application program.

In this section, the speed-up ratio and the factors affecting it are examined.

For the multiple-computer system presented in this paper, a cycle is the

[1]* Kober, R., "A Fast Communication Processor for the SMS Multimicroprocessor System", Second Symposium on Micro Architecture, M. Sami, Et.Al (Ed's), North-Holland Publ. Co. 1976, pp. 183-189.

time allowed to complete a write plus a read on the global shared-memory bus. During each cycle, a set of calculations is also performed by the individual processors. The physical sampling period is a function of the significant highest natural frequency of the system being simulated. The sampling period consists of several cycles, as it will normally require several cycles to perform the required calculations that must be completed during each sampling period. Because the total computation is performed by a repetitive sequence of cycles, the speed-up ratio is based on only one cycle.

Consider a multiple-computer system which has n individual processors and a total computation load of M tasks where a task is a self-contained portion of this load. Once a task is initiated it can be completed without the need for additional inputs.

The average computation time for one task is denoted by T_A. The average time for data exchange on the shared-memory bus per task with only global shared memory is denoted by T_C. The average time for data exchange on the shared-memory bus per task with both local and global shared memory, T_C', is given by

$$T_C' = k\, T_C$$

where k is the local shared memory factor ($0<k\leq 1$). A lower bound for k is 1/n.

If k=1, there is no local shared memory and shared variables are communicated only through a global shared memory. For k<1 the average time for data exchange on the shared-memory bus is reduced by the presence of the local shared memory.

The average processor utilization for computation, α, is given by

$$\alpha = \frac{T_A}{T_M} \qquad (0<\alpha\leq 1)$$

where T_M = the maximum time allowed for computation. Given the above parameters T_A, T_C', n, M, and α, the speed-up ratio for the multiple-computer system with distributed control, β_d, can be determined.

The execution time needed for sequential computation of M tasks is given by

$$T_S = M\,T_A$$

The parallel computation time for M tasks by a multiple-computer system with distributed control is

$$T_{P_d} = M\,T_C' + \frac{M}{n\alpha} \cdot T_A$$

Therefore, the speed-up factor β_d is given by

$$\beta_d = \frac{T_S}{T_{P_d}} = \frac{1}{\frac{1}{n\alpha} + \frac{T_c'}{T_A}}$$

The maximum speed-up factor β_d is given by

$$\lim_{n\to\infty} \frac{1}{\frac{1}{n\alpha} + \frac{T_C'}{T_A}} = \frac{T_A}{T_C'},$$

The speed-up ratio for the multiple-computer system without distributed control, $\beta_{\bar{d}}$, is given by

$$\beta_{\bar{d}} = \frac{MT_A}{T_D + MT_C' + \frac{M}{n\alpha}T_A}$$

where T_D = duration of control phase.

The speed-up ratio is improved by the factor, γ, with the use of distributed control.

$$\gamma = \frac{\beta_d}{\beta_{\bar{d}}} = 1 + \frac{T_D}{MT_C' + \frac{M}{n\alpha}T_A}$$

VI. SUMMARY

An architecture for a special-purpose multiprocessor has been presented. Limiting the use of this multiprocessor to a single application allows it to be optimized with respect to the use of critical system resources. This optimization consists of 1) distributing the system control functions to individual processors by implementing local real-time operating systems, 2) distributing copies of the system memory to each processor so that all system write operations are global and all system read operations are local and parallel, 3) scheduling the communications that occur on the synchronous data bus, and 4) partitioning the program tasks to minimize interprocessor communications. The resultant system has been shown to have a significant speed-up factor over a single processor system.