A distributed processing system for naval data communication networks

by WESLEY W. CHU and DAVID LEE
University of California
Los Angeles, California
and
BRANDON IFFLA
Naval Oceanic System Center
San Diego, California

INTRODUCTION

With the advent of microcomputers and computer networks, distributed processing (i.e., the sharing of computing by several processors with each processor assigned to perform a certain task) becomes technically and economically feasible. Such architecture also provides system expandability, system reconfiguration, and fault tolerant capability for the system. The performance of such a system is not only affected by its application but also by the system architecture.

One of the important problems that affects system performance is interprocess communication. It is intimately related with the task partition and assignment of various process modules and the system bus structure that provides the interprocess communications. In this paper we will first describe a new system architecture which consists of modules with identical bus interface connected by a unique broadcasting bus structure. In order to reduce bus interface cost, serial broadcasting busses are used. Next, we describe the simulation model for studying the performance of the proposed architecture. Finally, we use the known program characteristics and workload profile of a Naval data communication network as an example of the proposed architecture in order to study its feasibility and performance. The quantitative relationships among process and bus utilization, message delay, and task partitioning and assignments of the distributed processing system are presented. These results provide us with insight and understanding of the behavior of the distributed processing system.

THE DISTRIBUTED PROCESSING SYSTEM ARCHITECTURE

The proposed distributed processing system as shown in Figure 1 consists of three module types: processor, memory and input/output (I/O) modules. Modules may communicate with each other via a unique broadcasting bus structure. Each module has an identical bus interface consisting of a serial send bus and several receive busses. The send bus transmits data serially to all other modules in the system. The receiver receives data serially on its receive busses. For such a bus organization, a system with N modules requires N busses to communicate among the modules.

Every module has an identical hardware bus interface (BI) and a front end processor (FEP). The FEP handles data transfers and protocol between the bus and the module, coordinates data transfers, discriminates between message types, and handles bus transactions for all modules. The FEP of a processor module acts as an input/output controller for the main processor. When it is implemented as a microprocessor or as a bit-slice processor, it provides a reliable and inexpensive approach to handling data transmission and bus protocol. It also provides an inherent distributed system intelligence which can be programmed to control the allocation of hardware resources. A microprocessor is used as the main processor of the processor module.

Data enters the system via the I/O module. The I/O module may store the data temporarily in its own FEP local memory, or it may transfer data directly to any other module via the bus. Similarly, the FEP of a processor module may store data received from the system bus temporarily in its own local memory or transfer it directly to the main processor local memory. The FEP of the memory module may accept data temporarily in its own local memory or transfer data to the common bulk memory. Any module can initiate system output by transferring data to an I/O module.

The processor module

The processor module organization shown in Figure 2 assumes a microprocessor or bit slice processor implementation. The module consists of a main processor, an FEP, local memories, a Direct Memory Access (DMA) facility, a
Figure 1—A distributed processing system architecture

Figure 2—An organization of the processor module

bidirectional processor to processor interface and a system bus interface. The main processor executes the application program. The main processor local memory consists of read only memory (ROM) for program storage and random access memory (RAM) for data storage. The FEP has a small amount of local memory programmed to handle bus I/O and bus protocol. Local memory is dedicated to each main processor for program storage to alleviate contention by several processors accessing common memory, thereby reducing the inter-module communications. The FEP transfers data to and from the bus, as relayed to it by the system bus interface, and writes the data in the main processor local memory through the direct memory access. The reverse process occurs when data is read from the main processor local memory.

The DMA facility allows the FEP to store data in and retrieve from the RAM of the main processor. This interface ensures that data transfers are transparent to the interrupt and I/O control hardware of the main processor. The processor to processor communication interface allows word transfers to take place directly between the main processor and FEP without both accessing the main processor local memory. Either the main or the FEP can interrupt the other directly through their respective I/O facilities. The FEP receives or transmits data from or to system busses and provides asynchronous communications to the main processor, I/O control facilities, or to the main processor local memory.

The broadcasting bus structure

The proposed system may be viewed as a network consisting of different types of modules each having its own dedicated send bus which broadcasts information to all other modules. Further, each module can “listen” to the information from all the other modules in the system. Bus bandwidth affects the delay in inter-module communications; a serial broadcasting bus structure is used to both increase the bandwidth and reduce the bus and bus interface complexities.

Further, faults generated by hardware failures in such bus organizations are not likely to affect more than one bus, thus providing a graceful system degradation if software allows. The proposed bus structure can also be arranged to transmit parallel data thereby increasing the bandwidth for inter-module communications. However, system bus reliability decreases as the bus hardware complexity increases, and increasing the number of bus lines also increases the costs of the system bus interface. Hardware allocation schemes for bus hierarchies inherently become more complex as the number of busses increases. For real time applications, contention for a bus among modules is a severe problem when the contention time exceeds a specified time requirement for the system to respond to an external interrupt. Broadcasting bus organizations alleviate bus contention problems.

For the proposed bus organization, a system with N modules requires N busses to communicate among the modules. The system bus interface of each modules consists of N individual bus interfaces. Each bus interface (one per bus) decodes control signals from the FEP, performs serial-to-parallel data conversion, encodes and decodes character parity checks, and stores data for transmit or receive. Commercially available LSI components such as Universal Synchronous/Asynchronous Receiver/Transmitter (USART) or
Intel microcomputer (e.g. 8048) may be used for such interfaces.

Inter-module communications

Communication between modules is initiated by a module (called the source) transmitting a header, a character, or a block of data at a time. The bus interface of each receiving module of the system receives and temporarily stores the character of data and interrupts its FEP. The FEP of the receiving module honors the interrupt by resetting the interrupt facilities in the bus interface and then reading the identification of the bus request. The FEP of the receiver reads and decodes the destination address. The source begins transmitting data after receipt of an acknowledgment from each of the designated receivers. The receiving FEP disables the bus communication by not resetting the interrupt facility if the data is not addressed to it or if it does not desire to receive the data at that time. The receiving FEP records the time when the transfer request character was recognized. Since a receiver must send an acknowledgment honoring the transfer data request within a specified "time out" period, the source waits to receive an acknowledgment before it begins sending its block of data. All data communication on the bus is in block format. Asynchronous data communication requires DATA and ATTENTION lines for each bus. Synchronous data communication requires DATA, CLOCK, and ATTENTION lines for each bus (Figure 2). The receiving FEP takes one character of data at a time from the bus interface and stores the data temporarily in its own local memory or stores the data, via the DMA interface, directly at a predetermined memory location in the main processor local memory. After a receiver acknowledges the header character (which may consist of destination ID, word count, priority and message types), additional characters are transmitted by the source as specified by the word count of each message block.

In order to transfer data from the main processor to the bus output the main processor interrupts the FEP via the processor to processor interface. The data transfer flag for each receiver is stored in the RAM of the main processor. After the FEP acknowledges the interrupt, it reads the data transfer table if there is a request for a data transfer. The FEP initiates the output of that data block to the system bus interface. Transferring input data to its main processor is accomplished in a similar manner. Since program and local variables are stored in the local memory of each main processor, interprocess communication is greatly reduced.

Input/output module and memory module

Both the I/O module and the memory module (Figure 1) have an FEP for handling data transfers to and from the bus, and have identical system bus interface, FEP, and local memory. The memory interface in the memory module performs the function of interfacing the FEP with the bulk memory. The channel interface in the I/O module asynchronously multiplexes the channels and provides asynchronous communication with its FEP, I/O control, and interrupt facilities.

The I/O module interfaces the system with the user and with a variety of peripheral devices that include various hardware interfaces. The I/O module stores the data in its own local memory which queues the data for output to an I/O channel interface. In a similar manner, the FEP retrieves data from its local memory and sends the data to the bus. The interconnection of the FEP of the I/O module to the channel interface depends on the specific application and channel interface characteristics.

PERFORMANCE EVALUATION

In examining the performance of a distributed computer system, we are interested in the study of inter-module communications, processor and bus utilization, and response time and throughput of the system. Since these performances are program and application dependent, analytical models that can accurately predict the system performance are difficult to devise. An alternative method is to construct a prototype and measure these performance values. This approach not only requires the development of hardware prototypes, software system, data reduction and measurement facilities, but also encounters difficulties in evaluation of the system performance at different parameter values. We therefore use simulation techniques which enable us to use the actual program behavior and workload profile in studying system performance.

The simulation model

In order to provide a general yet accurate simulation model for our study the model is based on events. Events are generated at each processing module according to a given arrival distribution and enter into multi-level queues waiting their turn to be processed. The processing time of each task is known. In evaluating the performance of the system we are only interested in the task processing sequence, the task time and the bus occupancy time. The information of processor instructions and bus traffic contents is not required. As a result, the model does not require the detailed knowledge necessary for the applications program and system software. This greatly simplifies the simulation. The accuracy of the result depends on the accuracy of the estimates of processing times, bus occupancy times of various tasks, and the workload profile.

The simulation model for a module is shown in Figure 3. The bus occupancy time of a task can be estimated from the number of bits transmitted and the bus bandwidth. All the queues are of a multi-level queueing system, with entry level determined by the priority of the messages. The model also has facilities to gradually increase the message priority as a function of the amount of time since it entered the system. The simulation is event-driven. That is, it advances the clock to the next event which is to occur, rather than stepping
through every clock interval and checking for events. Thus
the time required for a simulation run depends on the num­
ber of events which occur during the simulated interval
rather than the time span of the simulated interval. The
internal unit of time is expressed in terms of bus clock time.
Inputs and outputs are represented in milliseconds.

A module can receive data simultaneously from several
busses, but can send data only on its designated send bus.
The number of received busses, the number of modules in
the system, as well as the buffer space can be easily varied
in the simulation.

Input parameters for the simulation are message arrival
rate, which corresponds to the satellite communications
channel bandwidth, effective processor execution time,
which reflects both instruction execution time and the power
of the instruction set, bus bandwidth, assignment of proc­
sessing tasks to the modules, and number of processing mod­
ules.

The performance measures used in the simulation are
utilization of the processors, busses, and peripheral devices,
delays at each module such as processing queues, sending
queues to the bus, receiving queues from busses, and mes­
sage response time.

The simulation is written in PL/C, Cornell University’s
compile-and-go version of PL/1. Because subroutine calls
have a high executed time overhead, no subroutines are
used. Integer variables were used whenever possible in
order to save execution time. The processing time required
on the IBM 360/91 for a typical simulation run ranges from
one to two minutes.

EXAMPLE FOR SIMULATION

The application selected for simulations was a communi­
cations processor (at a ground station) in a military satellite-
based communications network. This is mainly because rea­
sonable estimates of the computational requirements, work­
load profile, as well as program structure (program size,
execution sequence and times) were available from a pre­
vious study that was implemented with a military computer
(AN/UYK-20).1,2

The communications network shares a single half-duplex
channel of a satellite in stationary earth orbit. Reservation
time division multiplexing (statistical multiplexing)3,4 is used
for multiplexing the communications from all the stations.
There are two types of stations: master station and slave
station. The master station not only handles all the slave
station tasks but also it handles time slot reservation for all
the slave stations. In our investigation we shall study the
performance of the communication processor in a master
station. The basic time unit (time slot) is the time required
to transmit a 4,164 bit block. These time slots are grouped
into a frame which consists of 15 to 32 consecutive time

---

Figure 3—Simulation model for a module

From the collection of the Computer History Museum (www.computerhistory.org)
slots. The first time slot of each frame is for reservation of the remaining time slots in that frame. The master station makes the time slot assignments on the basis of requests received from all the other stations (slave stations). These requests, which indicate the priority and number of messages a slave station is waiting to transmit, are sent during designated assignment request time slots. An assignment permits a designated station to broadcast a message block to all other stations. The stations addressed by the header (part of the 4,164 bits) need to fully process the message. For our application, we always keep the channel busy. Therefore, dummy messages are transmitted (assigned by the master station) when there are not enough real messages to occupy all the available time slots. The amount of work involved in handling these dummy messages by the communication processor at the received station is similar to the handling of those messages that are forwarded to the other stations. Each block of data consists of a 16 bit cyclic redundancy check code. Error detection and retransmission are used for error control.

**Assumptions used in the simulation example**

In the simulation example, we assume the round trip propagation delay between the radio (ground station) to the satellite is 260 msec. All the peripheral devices are assumed to have constant service times: Disk=90 msec; Printer=200 msec (at 6000 lines/min.); Magnetic tape drive=150 msec. Processor execution time ratio is relative to a typical military minicomputer (AN/UYK-20) with instruction execution time of 1.5-2.3 msec. Execution time includes the effects of both instruction execution cycle time and the power of the instruction set. Each module has 16K bytes of buffer space available for bus traffic and can receive data from three busses (i.e., three other modules) simultaneously. A message fits exactly into one message block (4,160 bits including message header and cyclic redundancy check bits). Processing of a send message has a higher priority than the processing of a received message. This guarantees that send processing suffers minimum of delay, which assures that messages to be sent will be ready for transmission in their assigned time slots (as required by the network protocol) with a reasonable processing scheduling scheme. The bus bandwidth is assumed to be 50 kHz/bus.

**Task partition and assignment for the example**

The two main tasks of the communications processor at the master station are to perform send and receive message blocks. Their task flow sequence is shown in Figures 4 and 5. Additional tasks such as generation of network and station status reports for the operator, creation of checkpoint tapes for system recovery in case of failures, etc. occur relatively infrequently. For purposes of simplicity they are not included in the simulation.

The basic principle of a good assignment of processing tasks to processing modules is to balance the processing load and minimize the amount of inter-module communic-

**a) SEND A TIME SLOT RESERVATION MESSAGE**

- **INITIATE SEND PROCESSING**
- **GENERATE ASSIGNMENT LIST**
- **APPLY HEADER AND ERROR CONTROL CODE**
- **RADIO**
- **UPDATE MESSAGE STATUS RECORD (SENT OK)**

**b) SEND A TEXT MESSAGE**

- **INITIATE SEND PROCESSING**
- **RETRIEVE MESSAGE TEXT**
- **APPLY HEADER AND ERROR CONTROL CODE**
- **RADIO**
- **UPDATE MESSAGE STATUS RECORD (SENT OK)**

**DISK**

Figure 4—Sequence of processing tasks for handling a send message
a) RECEIVE AND STORE MESSAGE ON DISK

STORE TIME SLOT RESERVATION REQUEST

CREATE MESSAGE STATUS RECORD

UPDATE NETWORK STATUS RECORDS

STORE MESSAGE TEXT

QUEUE FOR OUTPUT

UPDATE MESSAGE STATUS RECORD (STORED OK)

DISK

b) OUTPUT MESSAGE TO OPERATOR

OUTPUT RETRIEVE PERIPHERAL DEVICE UPDATE MESSAGE QUEUE

MESSAGE TEXT FORMAT MESSAGE CONTROLLER STATUS RECORD MANAGER (DELIVERED OK)

PRINTER MAGNETIC TAPE DRIVE

DISK

Figure 5—Sequence of processing tasks for handling a receive message

tion. Assigning all the tasks to one module requires the least amount of inter-module communication but may result in long queueing delays due to processor overload. On the other hand, assigning each task to a module yields short queueing delay for the processors but requires a large amount of time for sending intermediate results between modules. Because of the inter-module communication, processing requirements are rather complicated and may vary from one application to another. We shall use the simulation model to compare the performance of two task assignments.

The first assignment consists of assigning the tasks to a five-processor module system as shown in Figure 6. The task for sending a message on the communication network is largely independent of the task for receiving a message from the network. To minimize inter-module communication between the send task and the receive task, they are assigned to separate processor modules. We assigned one module for status record updating and time slot reservations, one module to serve as peripheral device controllers, and one module for file management and system recovery.

The second assignment consists of assigning the tasks to a three-processor module system as shown in Figure 7. Both sending and receiving processing require access to status records, but usually there are more receive messages than send messages. In order to minimize the amount of inter-module communication, status record updating and time slot reservation operations are combined with the receive processor. System recovery and report generations are combined with the send processor. The file manager and disk controller are combined, along with the peripheral device controllers, into a single module.

Table I displays the performance comparison of the three-processor and five-processor module systems. We notice

<table>
<thead>
<tr>
<th>Number of processor modules</th>
<th>5</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual time to process a received message (msec)</td>
<td>1601</td>
<td>1510</td>
</tr>
<tr>
<td>Average queueing delay to receive a message (msec)</td>
<td>14</td>
<td>13</td>
</tr>
<tr>
<td>Total # of bits transmitted on buses to process one received message</td>
<td>13504</td>
<td>8736</td>
</tr>
</tbody>
</table>

The system is operated at the following condition: 59 percent of the received messages are addressed to the master station and require complete processing; Network data rate=9600 bits/sec.; Bus bandwidth=50kHz; processor speed equal to a minicomputer.
that the average delay in processing received messages is almost the same in both of these configurations. Due to less inter-module communication in the three-processor module system, less bus traffic is required to transmit the intermediate results and less processing time is required to prepare and interpret these inter-module communications. As a result, the three-processor module system yields better performance than that of the five-processor module system.

It is interesting to note that due to the excessive inter-module communication and the overhead in handling these

Figure 6—a five-processor module organization for the example.  
M = Memory requirements for program storage (in bytes)
inter-module communications in the five-processor module configuration, the throughput as well as response time of the three-processor module are better than that of the five-processor module system. Therefore, in the planning of a distributed computer system, using more processors does not necessarily always yield better throughput. In fact, one of the important problems in the planning of a distributed processing system is how to partition the tasks and optimally assign them to the processor modules such that they yield minimum interprocess communications and high system throughput, and yet satisfy the cost and response time requirements.

**Discussion of results**

Total delay is defined as the time difference for processing a message with and without any queuing delay. Total delay is not necessarily equal to the sum of all the queuing delays encountered in the processing of a message, since some processing (and delays) occur in parallel execution branches. Delays are shown only for the processing of received messages, since the processing of send messages has a higher delay allowance than that of the receive messages. Since the message arrival and departure process are deterministic and there is a long radio propagation delay, there is no interference between network cycles. And since the processing of received messages is completed before the processing of send messages begins, we only need to simulate one complete network cycle. We studied the simulation variations of the network input data rate and relative processor execution time with the three-processor module configuration. The system performance (with processor execution time equi-
Figure 9—Peripheral device delay for processing a receive message vs. network data rate (10^6 bits/sec). Note: mag. tape drive delay is not included in total because it always occurs at the same time but is shorter than the printer delay. Relative processor execution time = 1. Bus clock period = 20 msec (50 kHertz/bus).

For the relative processor execution time exceeding 5 (Intel 8080 has a relative processor execution time of approximately 4 in this application), the total delay for processing a receive message exceeded the maximum allowable time which is 218 msec for 19,200 bit/sec and 436 msec for 9,600 bits/sec. From Figure 12 we noticed that for the system with large relative processor execution times, the delay of a received message is dominated by the processor delay. The increase in link queueing delay with increasing processor execution times is due to the increases in the effective peripheral device service resulting from the reduction in processor speed. The decrease in disk queueing delay at high processor execution times is due to the stretchout of

* Based on approximately equal numbers of large blocks of 4,192 bits each (about 85 msec) and small blocks of 192 bits each (about 4 msec) are sent on the busses during the processing of messages.

For the relative processor execution time exceeding 5 (Intel 8080 has a relative processor execution time of approximately 4 in this application), the total delay for processing a receive message exceeded the maximum allowable time which is 218 msec for 19,200 bit/sec and 436 msec for 9,600 bits/sec. From Figure 12 we noticed that for the system with large relative processor execution times, the delay of a received message is dominated by the processor delay. The increase in link queueing delay with increasing processor execution times is due to the increases in the effective peripheral device service resulting from the reduction in processor speed. The decrease in disk queueing delay at high processor execution times is due to the stretchout of

* Based on approximately equal numbers of large blocks of 4,192 bits each (about 85 msec) and small blocks of 192 bits each (about 4 msec) are sent on the busses during the processing of messages.

Figure 10—Utilization of busses and bus receivers vs. network input data rate. The bus transmitter utilization is the same as the bus utilization. Bus clock period = 20 msec (50 kHertz/bus).

For the relative processor execution time exceeding 5 (Intel 8080 has a relative processor execution time of approximately 4 in this application), the total delay for processing a receive message exceeded the maximum allowable time which is 218 msec for 19,200 bit/sec and 436 msec for 9,600 bits/sec. From Figure 12 we noticed that for the system with large relative processor execution times, the delay of a received message is dominated by the processor delay. The increase in link queueing delay with increasing processor execution times is due to the increases in the effective peripheral device service resulting from the reduction in processor speed. The decrease in disk queueing delay at high processor execution times is due to the stretchout of

* Based on approximately equal numbers of large blocks of 4,192 bits each (about 85 msec) and small blocks of 192 bits each (about 4 msec) are sent on the busses during the processing of messages.

For the relative processor execution time exceeding 5 (Intel 8080 has a relative processor execution time of approximately 4 in this application), the total delay for processing a receive message exceeded the maximum allowable time which is 218 msec for 19,200 bit/sec and 436 msec for 9,600 bits/sec. From Figure 12 we noticed that for the system with large relative processor execution times, the delay of a received message is dominated by the processor delay. The increase in link queueing delay with increasing processor execution times is due to the increases in the effective peripheral device service resulting from the reduction in processor speed. The decrease in disk queueing delay at high processor execution times is due to the stretchout of

* Based on approximately equal numbers of large blocks of 4,192 bits each (about 85 msec) and small blocks of 192 bits each (about 4 msec) are sent on the busses during the processing of messages.
access request arrivals resulting from the increased processor queueing delay. The variation in bus delay to received messages is the result of specific timing relationships between the processing of send and receive messages that occur at certain processor rates, which is due to the variations of starting time for processing of send messages. The queueing delay at the Receive Processor for processing a receive message is approximately equal to that of a PP module (i.e., balanced delay). The delay from the PP module is due to the larger number of processing tasks, each of shorter duration that for the SP module. Processor utilization (Figure 14) is the overall utilization average for the whole network cycle. When the processor execution time is large, the demand for processing may be higher than 100 percent during part of the time. This yields a very large queueing delay.

With the three-processor module configuration with Intel 8080 type microprocessors, the system performance is adequate for handling of the example network with a data input rate of 9,600 bits/sec. Since three-processor modules are just adequate, it is very unlikely that a system with a two-processor module configuration could do the job satisfactorily. The response time for the two-processor module is likely to be much larger because the processing tasks cannot be easily divided into two equal parts. Consequently, additional delays will be caused by increases in contention between processors and by unbalanced processor loading.

Simulation results reveal that the serial broadcasting bus structure is viable. The bus delays for receive messages are lower than the processor delays for a relative processor execution time greater than 3 and are comparable to the processor delays for a relative execution time in the ranges between 1 to 3 (Figure 12). Bus delay can be reduced con-
siderably with a higher bandwidth bus. A much higher bus bandwidth interface with 2.5 MHz (50 times higher than the bus bandwidth used in the simulation) is available commercially.

We also noticed that increasing the network data rate causes a higher bus and processor utilization, while increasing relative processor execution times increases only the processor utilization.

CONCLUSIONS

Simulation study revealed that the proposed distributed processing architecture with three processor modules provides adequate processing capability for handling the Naval data communication network processing tasks. Further, the serial broadcasting bus structure provides sufficient bus bandwidth for the application.

We also found that task partition and task assignment to various processor modules has great influence on interprocess or intermodule communications and thus affect the system throughput. A good rule for optimal task partition and assignment is to balance the workload among the various modules and yet keep them as independent as possible so as to reduce inter-module communication. Excess inter-module communication not only requires excess bus bandwidth but also requires extra processing time for handling these inter-module communications. This degrades system throughput.

The low bus and processor utilization for the system indicates that our proposed distributed processing architecture would be useful for other applications as well. Distributed processing architecture not only is more flexible for system reconfiguration and system expansion, but it also provides greater fault tolerant capability. Because of these capabilities and its low implementation cost, distributed processing architecture should find its place in many future applications.

ACKNOWLEDGMENT

The authors wish to thank Mr. Howard Wong and Ms Dana Small of Naval Oceanic System Center, San Diego, and Joel Trimble of the Office of Naval Research, for their encouragement and support in carrying out this joint project between NOSC and UCLA.

REFERENCES
