Remote real-time simulation

by OMRI SERLIN
Control Data Corporation
Sunnyvale, California

and

ROBERT C. GERARD
The Boeing Company
Seattle, Washington

INTRODUCTION

Real-time simulation has traditionally been associated with small, committed digital computers located in close proximity to analog computers or simulation equipment. However, the recent development of multi-programmed hybrid systems showed clearly that exclusive control of the digital computer by one simulation job at a time is not only unnecessary, but is, in fact, economically indefensible, due to the abnormally high proportion of idle time inherent in simulation work. Users of such multiprogrammed systems have been operating successfully, two or more at a time, with elementary teletype or CRT-keyboard terminals as their only means of communication with the central computer. It is, therefore, reasonable to assume that, given the appropriate software and hardware elements, such terminals could be located substantial distances away from the central computer without compromising the level of interactive control of the simulation.

To test the validity of this assumption, an experiment has been carried out at the Boeing Company’s facilities in Seattle, using the Control Data® 6600 and 1700 computers. In the experiment, a real-time simulation of a supersonic aircraft is performed. A simplified “cockpit” and analog recording equipment are located some 10 air-miles away from the 6600, where the mathematical model of the aircraft is implemented. This experiment supplied the motivation for, and some of the results reported in this work. Many of the considerations, however, are universal in nature. In addition, the paper speculates on the desirable software and hardware features that will permit even higher levels of efficiency in running remote real-time simulations.

WHY REMOTE SIMULATION?

Many industrial organizations, government installations and universities possess large-scale computer installations to support their scientific batch and/or time-sharing needs. Remotely located simulation facilities can tap the power of these fast, sophisticated processors at a fraction of the cost of procuring and installing committed systems for specific needs. The central computer is likely to be fast enough so that the simulation can be coded in FORTRAN; the resulting advantages include faster debugging and easier modifications to accommodate model growth and change. The removal of the analog-digital proximity constraint permits better matching of sites to the differing hardware requirements of simulation gear and computer mainframe and peripherals.

PECULIARITIES OF REAL-TIME SIMULATION

Real-time simulation is characterized by computing tasks that are prompted by external events (“interrupt signals”), and that require the results of the computation to be made available to the external equipment either at the time of another external event or a specified time period after the occurrence of the initiating event. The external events can be periodic or non-periodic. In the latter case they are termed “asynchronous interrupts”. Typical periods (“frame times”) are of the order of 10 to 100 milliseconds. The amount of com-
puting time required in every such frame depends entirely on the complexity of the mathematical model and the speed of the processor. Using the CDC 6000 as an example, the required compute time for a typical aerospace simulation varies between 10% and 60% of the frame time.

These timing constraints distinguish the real-time simulation from time-sharing (TS) or remote-job-entry (RJE) systems where the constraints are one or two orders of magnitude less severe. Even more important is the fact that failure to meet a timing constraint is merely a nuisance in a TS or RJE environment, whereas the same effect in real-time simulation is often fatal in that it can cause irreparable damage to the simulation. (This is less true in pure digital simulations than in hybrid ones.) Tasks characterized by such fatal outcome of timing errors are often termed “time critical” or “constrained” computation.

The amount of data exchanged between the computer and the simulation equipment depends primarily on the division of tasks between these components of the system. Representative figures might be 10 to 25 inputs and 10 to 50 outputs at analog (≈15 bit) precision. Discrete (on-off) signals as well as pure digital quantities in varying sizes and numbers could also be exchanged periodically.

WHAT ARE THE UNKNOWNS?

By “remote” we mean that such simulation hardware as analog computers, AD conversion gear, cockpit mockups, and so forth, are located so far from the central computer, where the mathematical model of the aircraft or spacecraft is implemented, that standard cables cannot be used and some sort of distant communications equipment is required to carry signals between the computer and the remote facility.

The success of the remote simulation concept depends on several fundamental issues:

a. The Communications Facility

Standard switched or private line wide-band offerings of the common carriers, as well as private optical or cable communications links, are available. The reliability of these services is of interest. A related question is that of error handling, i.e., the recovery algorithm.

Of crucial importance (as will be seen later) is the speed of the communications link. In this respect, the question divides into two parts. First, how adequate is the available service? Secondly, how fast should it be?

b. The Remote Terminal

Since the central computer exchanges signals with the local communications interface only, it is completely unaware of the type and organization of the equipment at the remote site. Thus, these parameters are flexible and the designer can choose any one of several possible configurations. The question is, therefore, what is the best remote terminal configuration, and what criteria should be used to measure its adequacy?

c. Effect on Central Site

One of the basic assumptions of this work is that the primary mission of the central site is other than supporting real-time simulation. It is then extremely important to establish (in advance if possible) the effects of adding the simulation to the standard load, which is presumed to be a combination of batch, timesharing and remote-job-entry activities.

THE EXPERIMENTAL SYSTEM

Figure 1 is a schematic diagram of the system which was used to study some of the questions just posed. The following is a brief description of the components of the system.

a. The Communication Components

The wide-band communication link, provided by the telephone companies, is classified according to its band-
width. The most common wide-band services are the 48 KHz and the 240 KHz lines. The bandwidth places an upper limit on the data rate that can be transmitted over a given line. However, the actual transmission rate is determined by the data set.

Data sets are required at each end of the communication link. A data set (also known as modulator-demodulator or “modem” for short) is a special purpose, bi-directional analog-digital converter. Serial binary information, presented to the data set by the computer, control the modulation pattern of a continuous carrier signal* which is fed into the communication link. At the receiving end, the modulated analog wave shape is decoded into a series of on-off signals which are outputted from the data set serially. Data sets for wide-band service (about 20,000 bits/sec and over) are generally clocked. The clock rate controls the transmission speed. The three most common speeds are 40.8, 50, and 230.4 kilobits/sec.

The data set does not attempt any interpretation of the data passing through it. The Communications Controller, which interfaces the data set to the I/O facilities of the computer, is responsible for decoding the incoming data. In order to provide some facility for exchanging control information between the remote and local equipment, the controller can generally be expected to establish at least two modes. An IDLE mode indicates that no data is being received. In this mode special decoding circuits can search the incoming bit pattern for any number of special codes. The two most basic codes required are: (a) a code to signify the beginning of a data exchange, and (b) a code to signify that the remote station desires attention. The latter code can be wired to trigger an interrupt in the computer or a special status bit in the controller.

The data exchange is terminated when the computer simply does not accept (or supply) a word within the time required to receive (or transmit) the previous word. This, and several other features of the communications controller, are designed to achieve high-speed, efficient transfer of block data. The software implications of this organization are examined later.

The specific equipment used in our experiment included a 48 KHz line and a Bell*** 303 data set, clocked at 40.8 kilobits/sec.

b. The Computer Facilities

The central computer complex at which the work was done consists of two CDC 6600 computers, each with 131K (60 bit) words of memory. Briefly, the 6600 is an extremely powerful, large scale computer, incorporating a very fast 60-bit central processor and 10 peripheral processors, each with a private 4K 12-bit memory and a repertoire of simple arithmetic and I/O instructions. The peripheral equipment includes disks, tapes, graphic terminals, interactive typewriter terminals, and unit record equipment. Each 6600 is also equipped with a 4-port high speed communications controller, but normally only two RJE terminals are connected to a mainframe at a time. The AD link and simulation gear were installed at one such RJE terminal, consisting of a CDC 1700 with 16K of 1 microsecond, 16-bit memory, a disk operating system, a high speed communications controller and unit record equipment. More detailed information on these computers can be found in References 2 and 3.

c. Modification of Facilities

Certain revisions to the normal mode of operation had to be made so that testing of the remote real-time simulation concept could be carried out. We had determined that a realistic minimum frame time for RTS would be about 20 milliseconds. We also determined that the standard Export/Import (RJE) package could not be used to support the simulation. We, therefore, dedicated the high speed multiplexer (communications controller) and a peripheral processor to the real-time application. This meant that under operational conditions all other remote job entry work would have to be handled by the second 6600, or additional hardware would have to be purchased to allow RTS and RJE to run simultaneously on the same processor.

It was also determined that the real-time job should not be subject to storage relocation and, indeed, should be able to interrupt the storage relocation task in order to direct the CPU to the time critical application. This resulted in reducing the priority of the storage relocation program and assigning the highest CPU priority to the real-time job. It also resulted in assigning the real-time job to a memory area adjacent to the fixed-size system tables. This had the effect of automatically eliminating the need to relocate this job, without any modifications to the system.

Except for these modifications, which were of a minor nature, we were able to implement the real-time simulation within the framework of the current Boeing operating system. The ease with which we achieved this conversion is due to a large extent to the unique organization of the CDC 6000 computers, where the CPU is normally completely divorced from handling any I/O tasks. In a more conventionally-organized

---

*Completely digital transmission systems are available in some areas.

**Registered Trademark of AT&T.
computer, the real-time job will have to share the CPU with time-critical I/O operations (e.g., for avoiding lost data conditions on fast devices), and hence with non-interruptable system routines. In such an environment the conversion may be considerably more difficult.

d. The Linkage and Simulation Gear

In an attempt to keep the demonstration as simple as possible, while maintaining a degree of realism in the simulation, we chose to display outputs on strip chart recording equipment and to limit inputs to pitch and roll control from a simple "joy stick". Control Data provided a 1500-type interface between the 1700 remote terminal and the control and recording devices. The interface equipment included 16 AD and 16 DA channels with a resolution of 15-bits (including sign) and a maximum range of ±128 volts.

e. The Applications Program

The mathematical model program that was implemented on the 6600 was a fairly simple six degree of freedom rigid body representation of a supersonic aircraft in a high speed cruise condition. Debug and checkout time had to be held to a minimum, so the voluminous data normally used to describe engine and aerodynamic characteristics was eliminated, as was the autopilot. The simulation did include a complete set of equations of motion, linear aerodynamic coefficient buildup, all coordinate transformations, a nonlinear atmosphere, effects of wind and a complete set of flight control equations, including both longitudinal and lateral stability augmentation systems. Manual inputs were pitch and roll control. (We assumed fixed throttle and trim settings.) The outputs included all data normally displayed on the pilot's flight instrument panel. The simulation was quite typical of those used for piloted studies of airplane handling qualities, stability and control problems, or flight control system optimization.

The simulation along with the central control program (to be described shortly), the associated service routines (all coded in FORTRAN IV), system routines and I/O buffers required less than 12K of central memory. Execution time per frame was approximately 4 milliseconds.

SPEED AND RELIABILITY OF THE COMMUNICATION LINK

As indicated previously, standard wide-band facilities permit operation at 40.8, 50, and 230.4 kilobits/sec. Table 1 below translates these speeds to words/sec., for 12 and 15 bits per word. Maximum packing (into data-channel-width words) is assumed and incompletely packed words, as well as header (Begin Data) and trailer (cyclic code) words appended by the controller, are ignored. Hence actual rates can be expected to be slightly lower than those indicated in Table 1.

State-of-the-art ADC's convert at rates of between 50,000 and 250,000 15 bit words/sec. It is clear that the time to transmit a block of AD data over the communication link, even at the highest available speed, is much more significant than the time required to make the conversion. Since the computation at the central computer cannot begin until the AD data is in, and must be completed sufficiently ahead of the next frame, to allow time for the transmission of DA data, the speed of the communication link is a crucial factor in determining the permissible quantities of AD/DA data for any given frame time, or, conversely, the possible frame time for a given amount of data.

It is important to place this restriction in the proper perspective. In our specific case, the 2 AD, 14 DA and 10 control words used required about 8 milliseconds to transmit over the 40.8 kilobit/sec. line. Even allowing for some growth in the complexity of the central program, there is still enough time within the 20 millisecond frame to exchange some 20 additional quantities per frame. With the overlap option, which is described later, essentially all of the frame time could be used for transmission. Furthermore, a committed computer, comparable in cost to our remote terminal, would have possessed a much slower CPU and therefore provide substantially reduced performance in terms of the permissible computation per frame, or the permissible minimum frame time.

An interesting development recently reported is that of laser communication links offering speeds of up to 250,000 bits/sec. These devices have been rumored to be commercially available at a very attractive purchase price. They require line-of-sight clear path, of course; their range is quite limited; and their performance in adverse weather conditions is, apparently, as yet unproven. Nevertheless they may be attractive in some

### TABLE I—Analog Word Transmission Rates (Words/Sec.) at Various Line Speeds

<table>
<thead>
<tr>
<th>Bits/Sec.</th>
<th>12 bits/word</th>
<th>15 bits/word</th>
</tr>
</thead>
<tbody>
<tr>
<td>40.8 K</td>
<td>3400</td>
<td>2720</td>
</tr>
<tr>
<td>50 K</td>
<td>4166</td>
<td>3333</td>
</tr>
<tr>
<td>230.4 K</td>
<td>19,200</td>
<td>15,360</td>
</tr>
</tbody>
</table>

From the collection of the Computer History Museum (www.computerhistory.org)
situations. Microwave facilities as well as privately constructed cable links are also of interest, as are new common carriers' offerings of, for example, a 50 kilobits/sec. switched (dial-up) service and the promise of megabits/sec. links as a fall-out from the development of the phone-TV service (where you see as well as hear the person you call).

The question of speed is quite fundamental. If line speeds were compatible to the data rates sustained by local equipment, it would be possible (at least theoretically), through proper design of the communication equipment, to make the remote equipment appear to the computer identical to local equipment. The computer would then be able to function the remote equipment and interrogate its status, and the remote equipment would be able to send many varied interrupt requests to the computer. The cost of very-wide-band lines is likely to prevent this possibility from becoming a reality in the foreseeable future.

The particular controllers we used employ "cyclic code" check system. The transmitting station appends a special code word after the last word of the output blocks. The bits in this code depend in a known way on the bits in the data block. The receiving controller regenerates this code as the data is received, and then compares it to the code sent by the transmitting controller. A mismatch flags a status bit or creates an interrupt signal.

The recovery algorithm that we are using is quite simple. Whenever the receiving controller reports a cyclic error, the associated computer ignores the data block just received and instead uses the data received in the previous frame, which is still available in the unmaeked buffer. The error is also logged, and, if occurring at the central site, is reported to the remote together with its time of occurrence. This technique should be adequate for digital simulations. Its adequacy in a hybrid environment needs to be verified.

Our tests of line reliability under real-time conditions are incomplete at this writing. Since line conditions, load, weather and other factors affecting reliability differ considerably from one area to the next, it is doubtful that whatever results we obtain can be assumed to be valid everywhere. However, transmission error statistics collected during RJE activities at the Boeing Seattle facility indicate that the error rate is less than 1 in 10,000 transmitted blocks (.01%) between the hours of 2 PM and 8 PM, and is practically nil outside those hours. That rate includes some retransmission attempts; and since these retransmissions will not be attempted under real-time conditions, error rates can be expected to be no more than that reported above.

It should be noted, incidentally, that the error checking facilities built into the communications controllers, beside being highly sensitive (detecting essentially all short error bursts), also represent an actual improvement over the local AD link, where, usually, no error checking of any kind is performed.

HOW THE REMOTE SIMULATION IS ORGANIZED

Timing

Since the primary function of the timing mechanism ("real-time clock" or "interval timer") in a hybrid simulation is to trigger the AD conversion equipment, it would seem to be convenient to place the responsibility for timing the simulation with the remote terminal, particularly when the central computer is of a conventional type. In our experiment, however, the timing function is performed at the central site for two reasons. First, a 6000 peripheral processor easily performs this function. Since one is dedicated to the remote simulation task anyhow, any task added to it is essentially a zero-cost item and, in fact, represents a savings in eliminating the requirement for a hardware timer at the remote site. Secondly, and more significantly, by placing the timing function at the central site we automatically obtained an important debugging capability: namely, the ability to exercise the central-site CPU program under real-time conditions without using the remote terminal or the communication controller. An important benefit of this arrangement was our ability to run extensive throughput tests on the central site computer some two months before the central-site CPU control program and the mathematical model became operational. These tests are discussed later.

Sequence of events

Figure 2 shows the sequence of events occurring during every frame at both ends and their time relationships. In a more conventional computer, the functions performed in the 6000 Peripheral Processing Unit (PPU) will have to be allocated in part to a buffered data channel and in part to the CPU.

At the central site, the PPU detects the beginning of a new frame in its clock* routine and transmits a "Start-of-Frame" interrupt code to the remote. It then

* The clock referred to is the 6000 12-bit real-time clock (a standard feature) counting at 1 microsecond intervals. Our PPU maintains a 24-bit "software" clock by referring to the hardware clock periodically.
known to both computers. When not required by the real-time phase shift error is 2T, where T is the frame time. Standard switches the communications controller to the RE­

Figure 2—Sequence of events in a frame

This diagram shows the scheme of synchronized communication employed. A Start-of-Frame interrupt from the 6600 triggers a sequence of data exchange in a prearranged order known to both computers. When not required by the real-time simulation, the central and remote CPU’s are available for batch or other work.

Not shown is the option of transmitting to the remote previous (rather than current) frame data, to achieve a high degree of I/O—CPU overlap in the central computer. In this option, the phase shift error is 2T, where T is the frame time. Standard extrapolation techniques may be used to compensate for this error.

switches the communications controller to the RE­CEIVE mode in preparation for the expected AD data from the remote. When this data begins to arrive, the PPU accepts it into a temporary buffer in its own memory. After the correct number of words have been received, the PPU transfers them to Central Memory (CM) and requests the CPU from the monitor. When the CPU program is finished, it signals the PPU, and the PPU transfers the block of output (DA) data just generated by the CPU program to its own memory.**

The PPU then switches the controller to TRANSMIT, transmits the output block, and returns to watch the clock for the beginning of the next frame.

The computer at the remote end, with its communications controller at IDLE, waits for the Start-of-Frame interrupt code from the central computer. The receipt of this code interrupts normal processing on the remote computer and transfers control to an interrupt routine. This routine pulses the CONVERT com­mand for the DAC’s (digital-to-analog converters), and switches the SH (sample/hold) circuits to HOLD. It then inputs the AD values (a buffered channel operation can be used) and packs the data in a format which will result in the most efficient transmission, i.e., the least number of unused bits. This is particularly important when using the 40.8 K bits/sec. facility, since at this relatively low speed, data transmission can occupy the major portion of the frame time. The routine then releases the SH’s, switches its controller to TRANSMIT, initiates a buffered data channel output operation, and returns control to the operating system.

The end-of-operation interrupt from this channel calls another routine. This routine switches the controller to RECEIVE, in anticipation of DA data to be sent by the central computer, and initiates a buffered input operation, after which control is again returned to the system.

The data channel commences input only when the Begin Data code arrives. This time, the end-of-operation interrupt, which occurs after the last data word is received, schedules a routine which unpacks the DA data just received and loads it into the first ranks (buffer registers) of the DAC’s, in preparation for the next Start-of-Frame interrupt code, which will cause actual DA conversion. The controller reverts to IDLE automatically when the reception is complete. The remote now waits for the next Start-of-Frame interrupt code.

Synchronization and central program organization

The remote communication equipment imposes certain restrictions on the method of communication between the two computers. It is important to recognize these restrictions, since they are essentially independent of the type of computers involved.

1. No control, status, or parity information is carried within the data words or block. This is done in the interest of speed.
2. Control information can be exchanged only when data is not.
3. Typically, only one control (interrupt) code is available to obtain the other computer’s attention.
4. The controller does not know the size of the data blocks.

The practical implications of these restrictions are that the computer cannot function the remote equipment, that the remote equipment cannot interrupt the central computer with a priority structure and that both computers must be aware of the size of the data blocks exchanged.

** Optionally, the PPU can be made to transmit data from the previous frame, overlapping the transmission with CPU execution for the current frame. This permits increasing the volume of data exchanged (or, alternately, increasing the permissible CPU execution time) every frame, at the cost of increasing the phase shift error to twice the frame time.
Either solution is very expensive. The hardware solutions: use a communication link that will sustain the same data rate as local equipment, and design data sets and controllers to decode control information; or, dedicate a separate communication link for control and status. Either solution is very expensive.

Two logical* solutions to this impasse are possible. The one typically employed in RJE support-software, uses the interrupt code as an “attention getter”, followed by the exchange of a block of data of known length. This block contains either instructions on what is to follow, or requests for desired action. In order to prevent confusion, one computer is typically restricted to issuing instructions only, whereas the other issues requests only.

The second solution, which—as is evident from the preceding discussion dealing with the sequence of events—is the one that we use, is to periodically exchange data in known block sizes, and have the control or status information coded into known places in the data block. This technique is particularly suitable for real-time simulations, which are characterized primarily by the periodic exchange of blocks of data between the analog and digital domains.

In our case, the first word of both the input and output blocks is reserved for functions and status. Typically, the remote computer codes into its outgoing block mode control and other requests introduced by the remote user through his typewriter keyboard. The central computer uses the control word for mode acknowledge (to signify the acceptance of the remote user’s commands) and for error information, alerting him to errors occurring either in his central program or in the received data blocks.

Perhaps the most important outcome of this method of synchronization is the organization of the CM (Central Memory) program. Figure 3 shows the overall data flow and program organization in CM. The requests contained in the mode word received at the central computer often require further dialog with the PPU (somewhat like system action requests in more conventional computers); at other times, these requests need only be interpreted by the applications program (for example, a mode change from OPERATE to IC); and sometimes both system action and applications program cooperation is required. To relieve the user of the need to communicate with the PPU or the system, a central memory control program was introduced.

This central control program (coded entirely in FORTRAN) acts somewhat like a monitor or a supervisor as far as the applications program is concerned, but appears to be just another user’s job to the system’s (real) monitor. Rather than have the user communicate with the system and PPU through FORTRAN callable subroutines, as is normally the case in conventional real-time simulations, the user’s program is, formally, a subroutine of the control program.* The control program, which is called by the PPU at the start of every frame, examines the mode word in the input block just received. The control program initiates system or PPU actions, if required, before transferring control, with a minimum of fanfare, to the user.

The interface between the user’s program and the control program is in labeled COMMON, to which the user supplies arrays of scale factors and unscaled, floating output data, and from which he retrieves input data ready for unscaled floating computation, as well as mode information.

The resulting modular program organization not only frees the user from the drudgery of handling complicated, machine-dependent control functions so that he is able to concentrate on developing his mathematical model program, but also permits replacing one application program by another with a minimum of pain.

The periodic data exchange, by keeping the communication link alive at all times, permits the IC mode to become an important debugging tool, since information flow is dynamic but the central applications

* The hardware solutions: use a communication link that will sustain the same data rate as local equipment, and design data sets and controllers to decode control information; or, dedicate a separate communication link for control and status. Either solution is very expensive.

* Obviously, it is just as easy to require the user to execute a call to the control program as his first executable statement. Aside from psychological considerations, there seems to be no real advantage to doing so.
program is static. Furthermore, a static test mode is easily implemented without requiring special programs to be loaded in either at the remote or the central site. In this mode, blocks of bit patterns are circulated through the entire system and searched for errors when they arrive back at the originating computer. The integrity of the communication path, as well as the reliability of the AD equipment, can be easily checked in this mode.

Simulation control at the remote site

The remote user is presently provided with an elementary level of control and monitoring of his job. Through the keyboard of his remote computer, he is capable of entering commands of essentially two types:

1. Simulation Mode Control
2. On-line debugging

Mode control commands include the standard OPERATE, HOLD, and IC which are meaningful primarily to the applications program. In addition, two special modes are available: STATIC TEST, which, as mentioned previously, is a hardware test mode intended to check the integrity of the communications path; and STANDBY.

The user enters the STANDBY mode whenever he anticipates a relatively prolonged period of inactivity (say 15 minutes or more) because of the need to attend to the simulation gear, or simply to analyze the results of the last run and prepare for the next one.

The function of the STANDBY mode is to permit the central computer to minimize the idle resources dedicated to the simulation. Upon receipt of this command the CPU is released. The PPU program copies the contents of the core space occupied by the applications and control programs to a file on the disk. It then releases the core space. Then, the PPU program goes into a "periodic recall" state, in which the PPU is released back to the system and is only claimed again for a very short period at regular intervals (nominally every 10 seconds). When the remote user is ready to resume operation, he reinitializes the remote computer; the PPU program detects this and resumes continuous control of the PPU. It then reclaims the necessary memory space and copies to it the program image previously saved on the disk. If, in the meantime, memory has been occupied by other jobs, operator attention is requested via a flashing message on the operator's CRT station. Automatic rollout of batch jobs under these circumstances can be implemented.

For on-line debugging, the remote user can select any variable from his central memory program and direct it to appear on any output (DA) channel by typing in an appropriate command. This has been achieved in a fairly unsophisticated way. The user is required to arrange all his variables of interest in COMMON blocks, and order the blocks so that each variable can be referred to by an ordinal relative to the beginning of the first COMMON block. A pointer array is maintained (also in COMMON) with an entry for each output (DAC) channel available. The contents of these pointer cells are the ordinals of the variables which are to appear on the corresponding channels. After completing its computation for the current frame, the applications program fills its output buffer as instructed by the pointer array. The remote user simply manipulates the pointer array contents when he wants to change channel assignment.

A similar technique may be used to insert values into any desired variable. The remote station would probably transmit to the central computer the characters as entered by the user. The central computer will perform the conversion to binary quantities so that the remote computer, which, in general, can be expected to have a much smaller word-length, is not required to carry triple or quadruple-precision quantities. Digital values to be displayed on the remote computer's typewriter will be treated similarly.

In general it is clear that a much more sophisticated debugging facility is highly desirable, including a symbolic insert/delete facility, as well as the ability to manipulate source files (or any other files for that matter). These capabilities are currently fully developed for local simulations, via an interactive CRT-keyboard user's station. A number of possibilities of bringing these capabilities to the remote site exist.

The remote user is also permitted to manipulate on-line, the frame time and the number of AD/DA channels.

Real-time I/O

In almost every real-time simulation, it is desirable to input and output other than analog data during the simulation. This is done, for example, to provide some means of printing flight history data while the simulation is in progress, or, at least, shortly thereafter. Other uses for this capability are the accumulation of error history files and, on the input side, the collection of telemetry data, the reading of functional tables for function generation tasks, and so on.

Standard FORTRAN coded or binary I/O statements cannot be used, since the compiler, taking into account the multiprogramming environment in which the object code will operate, codes these so that control
is returned to the operating system until the requested I/O is completed. Fortunately, the 6000 FORTRAN includes as a standard feature buffer I/O statements which allow the user to continue his processing after initiating the operation. The user must, of course, be careful to check the status of the previous buffer operation before attempting a new one to the same device. In general, the user is expected to use circular (chained tail-to-head) buffers, so that when one segment is in the I/O process, the other is being manipulated by the program. By increasing the size of the buffer, the user can relax his I/O timing constraints, often to the point where he can share the system's device (typically a disk), rather than requiring a dedicated unit (tape or disk pack).

In our experiment (Figure 3) we used this technique for accumulating flight history, error logs, and timing statistics on the system's disk. The files generated in this way can be transmitted to the remote for printout after the simulation.

In the future it may be very desirable to obtain on-line printout at the remote site. This can be achieved by adding a few words at a time to the output block. However, the number of words per frame that can be transmitted is clearly limited by the speed of the communications facility.

Remote terminal organization

It is worth noting that the system configuration shown in Figure 1 was designed to support remote job entry (RJE) facility under control of a combined 6000/1700 software system termed Export/Import. This facility, which is still available, although not simultaneously with real-time operation, must be regarded as mandatory. It is hard to visualize a serious effort to develop a real-time application remotely, unless the capability for volume source statement insertion and volume printing is present.

In later phases of the development, such as the debugging phase, the ability to quickly insert or delete a few statements at a time, into or from a source file maintained at the central site, becomes more important. Here an interactive CRT station could become a quite useful adjunct to the remote terminal.

In general it is safe to say that a general purpose computer is almost mandatory at the remote site (as opposed to special purpose hardware), to support the many varied activities ancillary to the real-time simulation. These include, among others, servicing of the user's station (typewriter or CRT/keyboard), and support of a card reader and a line printer for the RJE facility. Admittedly, the remote computer need not be as powerful as the one we used. However, if one of the many available "mini computers" is selected, the question of software development for both the simulation tasks and the related activities should be carefully weighed against the possible saving in mainframe cost.

EFFECT ON THE CENTRAL SITE

The installation is administered primarily as a very-high-volume, very-high-throughput facility.* Some 1200 batch jobs, submitted locally and remotely, are processed in a typical day. For this reason it was considered important to establish the effects of adding the remote real-time simulation load to this system. The primary objective of the throughput tests was to obtain quantitative data relating to these effects.

Test plan

The installation maintains two types of standard test devices that enable it to measure the effects of proposed changes—such as modifications to the operating system or installation procedures—on the current load. One such device is the Job Mix Sample. It is a set of some 70 actual jobs whose processing requirements are based on the known characteristics of the current system load. The first shift part of the job mix consists primarily of small, fast jobs, while the 24 hour mix also contains several large jobs with relatively long execution times. The Job Mix Sample is updated about every six months to reflect the current job mix characteristics. In effect, the Job Mix Sample compresses a whole day's operation to a several hours' run, and enables the installation to conduct controlled experiments.

The second device is an event-oriented (discrete) Simulator. The Simulator accepts (a) a description of the important parameters of the computer (e.g., core, number and speed of tapes and unit record equipment, etc.); (b) a definition of the queue-selection and priority determination algorithms; and (c) statistical distributions of job requirements, categorized by user groups, remote or local terminals, tape or non-tape jobs, etc. Job requirement distributions include arrival time and required resources. The Simulator samples these distributions randomly to create a job mix, and monitors job throughput, resource utilization and turnaround times. The Simulator processes a 3 days' load in 30 CPU seconds. Its accuracy is reported to be ±5% on throughput and ±15% on turnaround time.

* The interactive terminals, operating under SHARER, a special sub-system of the operating system, are few and are served only during certain hours of the day.
TABLE II—6600 Throughput Test Conditions

<table>
<thead>
<tr>
<th>Condition No.</th>
<th>RTS Requirements</th>
<th>Export/Import Running</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Core CPU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>40 K&lt;sub&gt;8&lt;/sub&gt;</td>
<td>—</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>50 K&lt;sub&gt;8&lt;/sub&gt;</td>
<td>—</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>40 K&lt;sub&gt;8&lt;/sub&gt;</td>
<td>25%</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>40 K&lt;sub&gt;8&lt;/sub&gt;</td>
<td>40%</td>
<td>No</td>
</tr>
<tr>
<td>6</td>
<td>40 K&lt;sub&gt;8&lt;/sub&gt;</td>
<td>40%</td>
<td>Yes</td>
</tr>
<tr>
<td>7</td>
<td>20 K&lt;sub&gt;8&lt;/sub&gt;</td>
<td>25%</td>
<td>No</td>
</tr>
</tbody>
</table>

We used the Job Mix Sample to determine the effect that the real time application had on the batch processing workload of the central facility. We first obtained actual throughput time under normal operating conditions and again under various conditions with the real time job active. Data points were obtained for the first shift portion of the Job Mix Sample, as well as for the entire 24-hour period Sample. The conditions for which these throughput tests were run are shown in Table II.

**Test results**

We have defined throughput degradation as follows:

\[
\text{throughput degradation (percent)} = \left( \frac{t(N) - t(ref)}{t(ref)} \right) \times 100
\]

where: \(t(N)\) is the time required to run the \(N\)th job mix case and \(t(ref)\) is the time required to run the job mix under the current production system. Thus a 100% throughput degradation refers to the situation where, because of the real time job requirements, the job mix takes twice as long to run as it does under the current operation system. Table III summarizes the results of the Job Mix Sample throughput tests.

Data collected from the 6600 simulator program are plotted in Figure 4. This graph depicts the throughput degradation that can be expected as a function of, and because of, the core required by a real time job. The curve passes through zero degradation at a real time job core requirement of 10K<sub>8</sub> because this is the amount of core required to support the RJE in the current production system, and RTS and Export/Import do not run concurrently, as explained earlier. Several data points obtained from Job Mix Sample test cases are also shown in Figure 4. These include the results from case 2 and 3 as well as extrapolated data points based on the difference between cases 4 and 7.

It will be noted that the correlation between the simulator results and the results of 1st shift throughput tests is quite good while the 24 hour job mix throughput results lie considerably further from the simulator curve. The statistical representation of the 24 hour job mix is probably not nearly as good as that of the 1st shift mix. Thus the discrepancies in Figure 4 are caused by too small a sample of jobs in the 24 hour mix.

Figure 5 shows the throughput degradation resulting from the real time job CPU requirements. The curves are plotted by subtracting the degradation due to RTS core requirements from the total degradation observed during the Job Mix Sample tests. This assumes that total system degradation can be obtained by merely adding the effect shown in Figure 4 to that shown in Figure 5. Two sets of curves are shown in Figure 5. One is drawn under the assumption that the simulator results (Figure 4) are applicable while the other is drawn assuming that the Job Mix Sample test data points apply.

**Figure 4—Throughput decrease vs. RTS core requirements**

Since RTS replaces RJE packages requiring 10 K<sub>8</sub>, the curve passes through zero decrease at that point.
TABLE III—Results of 6600 Job Mix Sample Throughput Tests

<table>
<thead>
<tr>
<th>Test Condition No.</th>
<th>Run Time (minutes)</th>
<th>CPU Utilization</th>
<th>Throuput Degradation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1st Shift</td>
<td>24 hour Job mix</td>
<td>24 hour Job mix (Case B)</td>
</tr>
<tr>
<td>1</td>
<td>33.55</td>
<td>75.28</td>
<td>60.4%</td>
</tr>
<tr>
<td>2</td>
<td>43.00</td>
<td>94.50</td>
<td>50.1%</td>
</tr>
<tr>
<td>3</td>
<td>45.43</td>
<td>100.75</td>
<td>47.6%</td>
</tr>
<tr>
<td>4</td>
<td>47.50</td>
<td>113.00</td>
<td>73.1%</td>
</tr>
<tr>
<td>5</td>
<td>55.30</td>
<td>131.40</td>
<td>83.6%</td>
</tr>
<tr>
<td>6</td>
<td>61.80</td>
<td>142.22</td>
<td>78.5%</td>
</tr>
<tr>
<td>7</td>
<td>41.65</td>
<td>103.30</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Figure 5 is plotted for a particular configuration of the 6600 system; i.e., no Export/Import operation, RTS assigned a dedicated PP and RTS assigned 40K(8) memory locations (which leaves 300K(8) for batch work). The CPU time is also assumed to be required on a 20 millisecond interrupt basis. However, included are the results obtained from case 7. Since the real time job was assigned only 20K(8) for case 7, and there is good correlation between the results of case 7 and the curves of Figure 5, Figure 5 is probably applicable over a wide range of real time job core requirements.

Figure 6 shows the average total 6600 CPU usage (in percent) by both the batch jobs and the real time simulation while the Job Mix Sample tests were running. Ideally, if every job used the CPU only when no other job needed it (i.e., during the other jobs' I/O wait periods), then 100% CPU utilization could be attained. As this figure shows, while there has been a dramatic improvement in CPU utilization due to the introduction of RTS, there are still occasions when, because of I/O or memory conflicts, no job is ready to use the CPU.

One unexpected result of the throughput tests was the amount of CPU overhead we experienced in getting the real time simulation on and off the CPU. In theory the “on” delay is not accountable as real time job CPU time and the “off” delay, which is accountable, averages about 300 microseconds each time the real time job releases the CPU. The accountable CPU overhead measured during the throughput tests averaged 900 microseconds per frame. We have not been able to identify the source of this overhead.
Figure 7—Typical strip chart recordings from the simulation

Roll and pitch commands are generated by a “joystick” and transmitted to the central 6600. Other traces represent results of the 6600 computation transmitted back to the remote site.

Using the results

Knowing the extent to which a real time application will affect the batch throughput on a central facility is one thing, but determining whether the operation could tolerate the degradation is quite another. A host of variables such as “spare” capacity on the central processor, the percentage of the time during which the real time job is inactive (“STANDBY”), relative economies and flow time associated with alternate approaches, as well as importance of the real time application, make sweeping generalizations rather difficult. It is possible, however, to identify several situations where the remote application concept would be an attractive solution to a real time computing requirement.

A central facility operating at less than 85% or 90% capacity (critical resource capacity) is, of course, an excellent candidate for implementation of the remote simulation capability. Once a production facility is operating above that level, the relative merits of batch vs. real time work come into play; but several conditions could result in the real time job taking precedence. Some of these conditions might be:

1. Limited duration of the real time application
   a. to support a “crash” effort (e.g., short-deadline proposal)
   b. to conduct a one-shot series of tests
   c. to allow time for purchase of additional computing equipment.

2. Availability of “outside” facilities to run batch backlog created by the real time application.

3. Restricting the real time application to certain hours of the day.

Special test facilities or hybrid simulation labs that are currently in the planning stages can quite easily be structured to take advantage of the remote simulation concept, since remote terminal processors that can double as special purpose stand-alone digital computers are available. In a lab such as this, much of the real time application work would be accomplished using the remote terminal processor only, but the speed and power of a large central processor would be available for those applications that required additional computing capacity.

CONCLUSION

A successful demonstration of remote real-time simulation was carried out in January, 1970. The demonstration consisted of “flying” the airplane simulation discussed in this report from the developmental center (location of the 1700) in Seattle, Washington. The math model was implemented on a 6600 in the Boeing Renton facility and communication between the 1700 and 6600 was via 48 KHz lines. Each mode of operation discussed earlier was shown to be operational. Figure 7 is a typical set of strip chart recorder traces obtained during the demonstration. Traces of the pitch and roll commands as well as computed roll, pitch, sideslip angle and angle of attack shown. This technique is presently being considered as the primary simulation tool for possible forthcoming work such as the B-1 effort.

The remote simulation concept, therefore, appears to offer important new possibilities in the design of new simulation facilities and in the manner of utilization of existing ones.

ACKNOWLEDGMENTS

Messrs. A. Ayres and Dennis Robertson, of the Boeing Company, coded the mathematical model for the CDC 6600 central computer.
Mr. Leo J. Sullivan, assisted by Mrs. Roberta Toomer, both of Control Data Corporation, coded the software for the CDC 1700 remote terminal and the CDC 1500 analog-digital link. Mr. Sullivan and one of the authors (O. Serlin) performed the actual checkout of the system.

Mr. Robert Betz, of Boeing, performed the throughput tests on the 6600 and assisted in analyzing the results.

Of the authors, O. Serlin is responsible for the conceptual design of the software, as well as the coding of the 6600 central control program, operating system modifications, and PPU program. R. C. Gerard planned and analyzed the throughput testing experiments, conducted the demonstration, and coordinated the Boeing Company’s efforts on this project.

Many other persons contributed to the project in other than the technical areas. Among these, Messrs. John Madden, of Boeing, and Tim W. Towey, of Control Data, deserve special recognition.

REFERENCES

1 M S FINEBERG O SERLIN
   Multiprogramming for hybrid computation
   Proc AFIPS FJCC 1967
2 J E THORNTON
   Parallel operation in the Control Data 6600
   Proc AFIPS FJCC 1964 Volume II
3 Control Data® Publication No. 60152900
   31700 Computer System Manual
4 J R BAIRD
   An optical data link for remote computer terminals
   Datamation January 1970