Interconnecting processors is one approach to organizing a computer facility to better serve its users.

The objective of such system organization is to reduce the elapsed time a job resides in the system (turnaround time) while simultaneously increasing the workload the equipment can handle (throughput.)

Alternative philosophies of multiprocessing are discussed and, in particular, a concept which enables coupling an IBM 7090 *and an* IBM 7040 *to meet this objective. In this system the smaller machine performs supervisory and input-output functions while the larger one performs program assembly and computation.*

A directly coupled multiprocessing system

by E. C. Smith, Jr.

We will be concerned with a general computer center which processes, primarily, problems originating with engineers and scientists in the normal course of their work.

Since the computational aspects of such problems can often be expressed algebraically, they often reach the computer center in the form of a detailed algorithm or program in an algebraic language, such as FORTRAN. The job of the center is to perform the algorithm and return computational results. If the program does not perform as desired, it may be changed and resubmitted to the computer center several times so that the checkout or debug phase of the computation must be considered.

Hence, in order to best fulfill its purpose, the computer center must perform and return each job as rapidly as possible. The elapsed time required to perform this cycle is commonly referred to as "turnaround" time. Once the user has accepted the constraints of specified programming languages and organizational procedures, his primary concern for better computational service is the reduction of turnaround time. He has, of course, a concomitant concern for the cost of this service. Consequently, if for example, FORTRAN functioning under the IBSYS OPERATING SYSTEM[1] dictates his only programming mode, then he asks only for better turnaround time constrained by reasonable cost. Of course, over-all problem solution may be considerably aided by

other means of problem and algorithmic statement, including such things as graphic input/output, conversational (real-time) coding and calculation, etc. However, that is not of concern in this paper.

Without the dominant constraint of cost one could duplicate equipment and consulting or service personnel so that each client could receive immediate service. Hence, the pertinent question is one or both of:

- How to provide better service with a given set of equipment and personnel.
- How to provide better service with a given amount of money.

These are organizational questions—organization of equipment and organization of people. Moreover, the first is a generalized statement of the question of "throughput"; the question of the amount of work that can be performed by a given set of resources.

Since a certain amount of work must be performed on each job between the time it leaves the problem originator and returns to him, the system engineer's job is to design a better system by an effort in three directions:

- Elimination of unnecessary work.
- Performance of each step as fast as possible.
- Performance of as many steps simultaneously as possible.

Multiprocessing is one approach to the latter concept of parallelism.

parallelism

Simultaneous execution of two or more functions within a computer has been done for some time. Actually, the IBM 704 was designed so that in some instances an instruction is obtained from memory before the execution of the preceding instruction is completed. The "look ahead" feature of the IBM 7030 (STRETCH) accomplishes parallelism of a similar nature in a much more sophisticated way.

On another level, the data channels on the IBM 709 allow computation to proceed in parallel with the transmittal of information into and out of core memory. This is carried much further in the IBM 7909 data channel on the IBM 7090/94. Here, a stored program of instructions is actually interpreted and executed in the data channel in parallel with interpretation and execution of the main program by the central processing unit.

More generally, the common "peripheral operations" of the transfer of information from card to tape and from tape to printer or punch usually proceed in parallel with the execution of programs by the main computers in an installation. Furthermore, the transmittal of information from the user to the computer room and back to the client proceeds in parallel with processing of other information.

multiprocessing

In this paper, *multiprocessing* will be used in the following sense: multiprocessing exists only when two or more processing units, each capable of interpreting and executing its own stored

program, operate *simultaneously on the same problem.* Furthermore, during such processing (although not necessarily constantly) there must be a transfer of information between the processing units (or their primary working memories) at or near memory access speeds. This communication is automatically satisfied if the main memories are not distinct, as is the case for the 7909 and the 7090/94. Thus, of the examples given in the previous section, only the 7909 data channel on the 7090/94 would be called multiprocessing.

Consider now two general purpose computers. In order to be specific, let them be an IBM 7040 and IBM 7090 although other pairs might be considered, such as the IBM 7044 and IBM 7094 II. Sometimes we may refer to them as the *input/output processor* and the *computational* processor. They may be connected together in one or both of two ways:

- A shared bulk (disk or drum) file.
- A direct data path between core memories.

When so connected, the resulting complex may be operated in a bewildering variety of ways. The result may or may not satisfy our definition of a multiprocessing system. The two processors may or may not work on the same problem simultaneously. Indeed, it may be that any given problem is submitted to only one of the two processors, and their connection is only in the sense that they both have access to the same bulk file of static reference information. Since this operating mode does not patently address itself to the problems of turnaround and throughput in a manner different from the duplication of equipment, we shall disregard it and consider only operating modes which may apply *both* computers to each problem.

At the other extreme, one can consider an operating mode in which both processors simultaneously work on the object program calculation. For example, one might be evaluating transcendental functions for the use of the other in integrating a differential equation. This may be considered as one of the "purest" forms of multiprocessing. With the present state of the art, however, its accomplishment requires great amounts of human analytical effort to suitably divide the over-all computational algorithm into parts for parallel performance. Most of our experience is in analyzing serial rather than parallel processes. Automatic procedures for logical algorithmic division into parallel parts remain to be developed and form today a fruitful subject for research.

function allocation

How, then, can we apply both the 7040 and 7090 to the same problems in a profitable manner? Since we choose not to divide the computational (arithmetic) functions between the processors, we shall let one of them (the 7090) perform all of these. The remaining problem handling and system overhead functions may be shared by the 7040, the 7090 and other system components (transmission lines, mail clerks, etc.). Since these functions have nothing whatsoever to do with the arithmetic algorithm involved,

they are brought into existence by the system and, hopefully, are subject to control by the system designer. These overhead functions include such diverse things as problem transmission from one location to another, collection of problem parts (subprograms) into a single file, scheduling, printing, accounting, conversion between binary and BCD and Hollerith codes, servicing physical input/output units, etc.

Again, we have a problem of organization. One must determine where in the complex each overhead function is to be performed and how much control over its functions each processor is to have. The choice of the processors themselves and the allocation of functions between them should be made on a criterion of best service per dollar. Unfortunately, this is greatly complicated by questions of existing legacies of programs, costs of converting programming systems, etc. In this paper we can only discuss functional allocation without full evaluation of attendant costs.

shared file concept

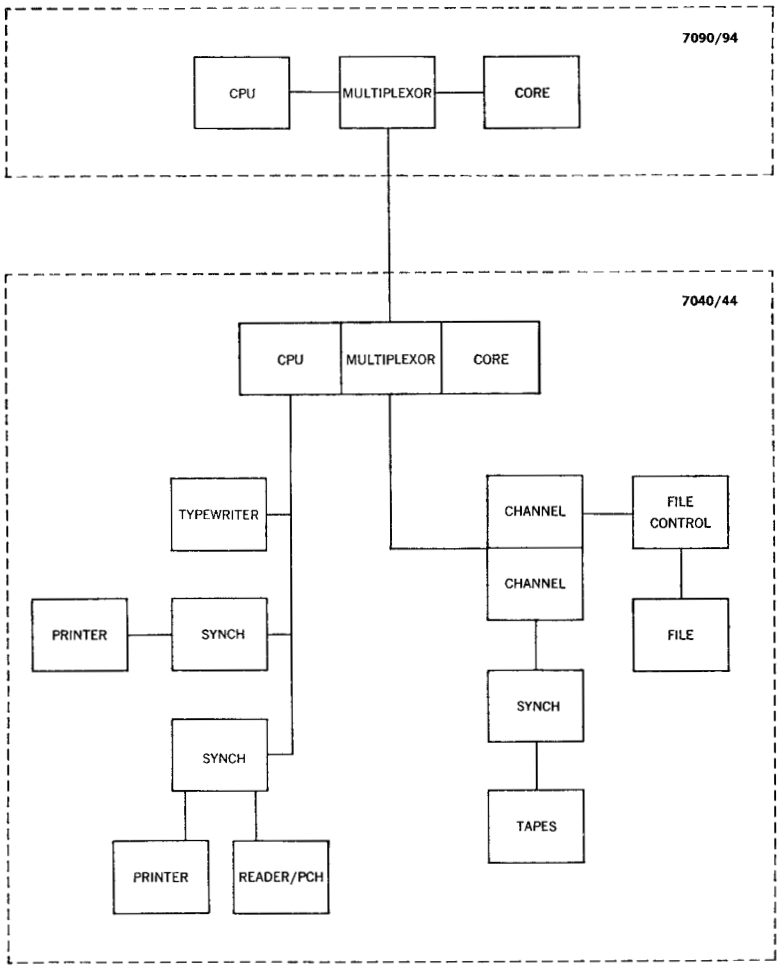
One philsosphy of system organization proposes that each processor be virtually independent of the other and have its own set of functions to perform when and as it alone chooses. The 7040 accepts system input and places it on the bulk file in a common (BCD) code. The 7090 selects its work from the file and returns results to the file. The 7040 then distributes (prints) final output from the file. More specifically, the computational processor may operate under a file oriented version of the standard IBSYS OPERATING SYSTEM. The I/O processor then serves merely the now very common peripheral functions of placing each job into a form and position accessible by the 7090 and communicating computed results back to the human world in printed and/or punched form. This, then, is the shared file system concept, sometimes referred to as the indirectly coupled system. An expanded form of this concept is discussed in Reference 7.

Since both processors in the shared file system do not simultaneously work on the same job, the system does not satisfy our definition of a multiprocessing system. Indeed, fundamentally the system simply attaches directly to the 7090 (via the file) some of the overall problem and result transmission functions of the global computer center system. This is not to deprecate the system, but to point out a philosophical distinction. Perhaps the term *multicomputer* is a suitable generic for this case.

directly coupled system

Another philosophy of system organization is that every overhead function that can possibly be taken from the 7090 and put into the 7040 be so placed. This implies, among other things, that the 7040 handles all (system and object program) input/output and that the 7090 operates as an arithmetic "slave" under the complete control of the 7040. This gives rise to the purest form of the directly coupled system concept. Clearly, a continuum of variants between these philosophies may exist and one's specific choice is based upon profitability. We shall describe a directly coupled system which places many but not all of the system overhead functions into the 7040.

Figure 1 Directly coupled multiprocessing system



Consider now an equipment configuration for the directly coupled system which consists of a 7040 with one disk channel, one tape channel, two printers, a card reader and a card punch together on one channel, and an independent connection to the core memory of a 7090. The 7090 has *no* other input or output facility. Both processors have 32,768-word magnetic core memories. The configuration is shown schematically in Figure 1. In order to effect the interconnection, both processors need to be modified. Each needs the ability to trap the other. That is, each needs to be able to interrupt the other and cause it to start executing a predetermined program at a specific location. The 7040 should also be trapped whenever the 7090 halts processing. This would normally occur only upon an error in a 7090 program, and would allow the 7040 to automatically insitute recovery procedures. A modification of the TRANSMIT instruction in the 7040 can effect the movement of a block of data between the two core memories without the use of what is now considered as common

channel logic. This gives the 7040 complete ability to load, start and monitor the processing function of the 7090. As we shall see, it also gives the 7040 the ability to perform all input or output operations required by the 7090. Consequently, it is appropriate to also call the 7040 the monitor processor.

The job input source to the system is the single card reader. The input processing for a job is handled by one of several subprograms called into the 7040 memory by the main supervisory program.

These programs effect the actual reading of each card and may do some pre-editing and packing by elimination of multiple blanks, etc. The input information is then accumulated into 460 word buffers, each of which is written onto one disk track by other subprograms called into action by the main supervisory program.

Most likely the subprograms activated by the supervisory program in order to perform functions such as those above would reside permanently on the 1301 disk unit and be called into action at the appropriate time. Each remains in core until replaced by an active program for some other function. Consequently, a subprogram with high usage would maintain almost permanent residency in core.

An important subprogram would be that which actually serves the disk file. In addition to effecting data transfer between the file and core memory, it should schedule the order in which disk accesses are made in order to reduce lost time.

As each job is stored on the disk file, notice of this may be made on the 7040 console typewriter. At this time the 7040 supervisory program may examine the queue of jobs awaiting processing by the 7090 and insert the new job into that queue at an appropriate place according to its priority and tape mounting requirements. This effects dynamic scheduling of jobs for the 7090 in order to give express service to high priority work and to minimize time the 7090 might be idle because required special data tapes are not ready. A wide variety of scheduling algorithms is feasible. The one chosen would probably depend upon individual installation requirements. The important point is that tape mounting messages are supplied to the machine operator by the 7040 sufficiently in advance of each tape's usage in order to allow it to be mounted and ready when called upon. Note that tape mounting messages need not be printed at the same time each job is actually scheduled for the 7090, but more likely at a later time according to mounting requirements.

The 7090 processes one job at a time, as it normally does today. This, of course, consists of program assembly or compilation or the running of an object program. In the latter case a system program loader (such as IBLDR in the IBJOB processor under IBSYS) is called into action.[4] The assembler, compiler or loader is called by the 7040 into the 7090. The 7904 data channel on the 7040 may be modified so that the 7040 can issue the necessary

commands and orders to initiate reading from the file and have the data flow from the file via the multiplexors of both processors directly into the 7302 core memory of the 7090. A better scheme might be to have the 7040 supervisory program look ahead sufficiently to read each required segment of systems program from the file into its core memory before it is required by the 7090. When required, it may be sent to the core memory of the 7090 by use of the 7040 TRANSMIT instruction at a rate of 16 microseconds per word. This is considerably faster than the rate at which information may be read directly from the disk file.

During processing, the 7090 intermittently interrupts the 7040 central processing unit with requests for data input or output. It is assumed that a modified form of the present day INPUT OUTPUT CONTROL SYSTEM (IOCS) resides in the 7040 for service to jobs being processed on the 7090. Linkage to this IOCS and other general supervisory routines in the 7040 is made by a minimal supervisory code which permanently resides in the 7090 core. A job being processed by the 7090 may call upon any tape or disk I/O unit attached to the 7040. Such calls have the highest priority of all system I/O calls but may still have to wait for service because of competing I/O activity.

When the 7090 computation of a job has been completed, its results then reside upon either the bulk file or magnetic tapes or possibly both. The 7040 console typewriter writes a line of information to log out each job. Final printing or punching of results may not have started, but the system maintains requests for such service and the 7040 performs them when possible. The 7040 supervisory program also schedules job output to the two printers and the card punch. The scheduling algorithm would consider both job priority and paper conditions in each printer. The supervisor must remember what kind of paper is in each printer and alert the machine operator by a typed message if a forms change is required.

Clearly, the 7040 is multiprogrammed but the 7090 is not. Assemblers, compilers and object programs do not have to operate in a multiprogrammed fashion. The only programs that do are system programs such as card-to-disk or disk-to-printer data transfer routines, input/output control, etc. Maximum system control resides in the 7040 since the computational processor works on the jobs only when and as directed by the monitor processor. Furthermore, since the monitor processor performs I/O, it thereby controls all of the input and output operations of the entire system.

input and output

It is the method of handling the input/output for the 7090 that makes the directly coupled system concept unique. Actually, it is but another step in the apparent progression of making the physical input/output control units more and more self sufficient. Here, the 7040 acts as an I/O control unit for the 7090 (as well as serving other functions). Like the 7909 data channel it interprets its own stored program, but unlike the 7909 it has its own memory for its program and for data buffering.

Fundamentally, when the 7090 makes an I/O call, it merely wishes to send a block of data to the other processor or receive a block from it. Parameters associated with this call must be transmitted to the 7040. It is possible for the 7090 to trap the 7040 and then sit idle while the 7040 obtains the calling sequence and effects the desired action. Upon completion of the call the 7040 traps the 7090, which allows it to resume computation. This can be done with a very small permanent program in the 7090.

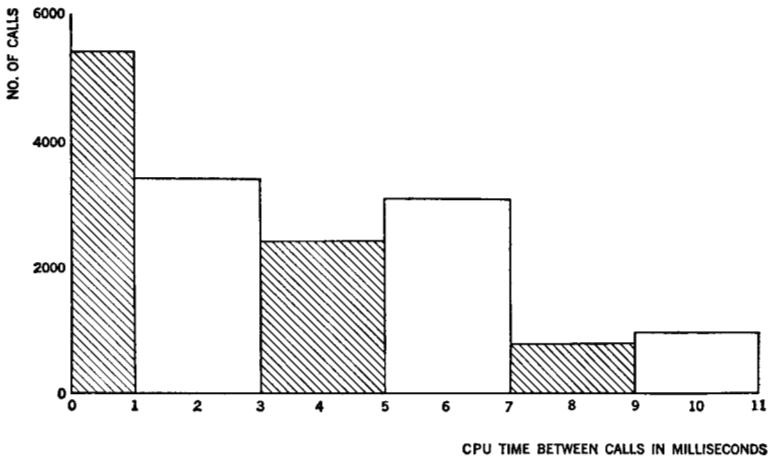
One restriction imposed by this scheme becomes apparent. Since all buffers and buffer pools are maintained in the 7040 memory, each compiler, object program, etc., which operates in the 7090 must allocate working storage into which data is moved from the buffers in the 7040 upon each call for input. That is, such data cannot be examined and processed while it still resides in system (IOCS) buffer locations. Even this could be avoided if one were willing to create a double buffer system with large buffers in the 7040 and fewer, smaller ones in the 7090. Then a good portion of an IOCS program would have to reside in the 7090, but all I/O select and trap supervision would remain in the 7040. This alternative is also attractive because the 7090 can interpret the calling sequence faster than the 7040 can. The disadvantage is that the core storage in the 7090 required for its IOCS program is now not available to be used by the object program, compiler program, etc.

If, as is assumed, one is further constrained to use a version of the IBJOB PROCESSOR, it is probably expedient to retain in the 7090 considerably more monitor code than indicated above. Even if as many as 2000 memory cells are retained for monitor program, there is a positive gain in memory space available to the object program in the 7090, when compared with a 7090 operating under IBSYS today. Furthermore, there is the possibility of larger buffers and buffer pools since these reside in another core memory. This alone implies fewer idle periods forced by incomplete physical I/O actions because it allows a more effective "look ahead" for data input and the temporary storage of more information in process of being written out.

Note that, despite appearances, this does not make a single channel machine out of the 7090. Several physical I/O channels may operate simultaneously on the 7040. Data transfer between the two memories functions logically like a move of a block of information between an IOCS buffer and working storage. Such a move in the 7090 is normally performed by a small program loop which requires 13.08 microseconds to move each word. The direct coupling moves a word between the core memories in 16 microseconds.

With this mode of handling I/O calls from the 7090, it is conceivable that the 7090 object program could make a second I/O call before the 7040 had finished resetting buffer pointers, etc., for the first call. It becomes important, therefore, to study the burst patterns of computation time for processing performed

Figure 2 Frequency distribution of inner-call times



between consecutive calls for I/O action for various types of computing jobs. One study of a very limited number of FORTRAN jobs produced the frequency distribution shown in Figure 2. It appears that this phenomenon will not degrade the performance of the entire system. Perhaps the importance of this particular work is to emphasize that very little is known about the I/O activity within the "typical" computing job, and much more needs to be known as more complex systems are developed.

system variation

System balance is a critical factor in the design of a multiprocessing system. The 7040 appears to be fast enough to serve well the individual I/O calls made by the 7090. It would not be fast enough to serve a 7094 II. In that case the smaller processor should be replaced by a 7044. However, when balanced to respond quickly enough to the I/O calls of the computational processor, it appears that the I/O processor may be busy less than half of the time. One now has the opportunity to remove more work from the 7090 to increase its throughput even more. Furthermore, one has a system design with many of the features required in order to attach remotely located terminals.

In order to automatically service remotely located terminals for job submittal and presentation of computed results, the system must be capable of handling many interruptions, asynchronously received. It must also have the time and memory required to sort and merge the various information segments to be transmitted from and to various terminals. The 7040, when programmed to operate much as described above, satisfies these basic requirements. Terminals may, therefore, be attached to the 7040 by means of a suitable data exchange without requiring further modification of the system's compilers, etc.

When a program in a source language is submitted to the system, the I/O processor might pre-edit the program to catch most of the bothersome clerical errors. This should significantly reduce the number of attempted compilations per job, and thus reduce the amount of work the 7090 must perform to service the

same over-all workload. However, since this diagnostic function is performed by present day assemblers and compilers, it would be duplicated on the 7090 when each job did reach it. Consequently, pre-editing on the 7040 should probably not be done unless an assembler or compiler which assumes such pre-processing is written especially for the system.

In the general system flow as described above, the 7090 still performs much work on each job prior to actual compilation or computation. This preparation consists of such things as the separation of comments from instructions to be compiled, consolidation of the various subjobs and subroutines which comprise the job, deletion or replacement of sections of code, conversion from relocatable to absolute binary format, and other functions of the general system loader. One might consider doing much of this work in the 7040 well in advance of the time of processing on the 7090.

Conceivably, one could rework the 7040 loader program, 7040 IBLDR, to perform some of this job preparation. In the directly coupled system, however, the 7040 operates in a highly multiprogrammed fashion. This has profound implications upon usage of memory space, program interruptability, etc. As a consequence, the 7040 IBLDR program would have to undergo major revision. The advantage of doing this would be to save 7090 machine time which is normally required for loading, because these functions would already have been accomplished well in advance by the 7040. The disadvantage of a major reprogramming effort can be avoided by maintaining a slightly modified version of the 7090 IBLDR, using it much as it is today, and omitting the preparatory step in the 7040 as suggested above.

One system feature that would be most desirable would be the capability to process current 7090 programs in the directly coupled system. Strictly speaking this is not a variant of the system described, but clearly would be a most important addition during a system conversion period. It is possible to modify the 7090 so that an attempt to execute an instruction that would normally refer to its data channels would effect a trap of the 7040. By suitable programming the 7040 could simulate the action of the data channels on the 7090 and, thus, allow most historical programs to run on the new system.

summary

When compared with a single processor mode of operation, the directly coupled multiprocessing system concept attacks the following:

- Reduction of lost time due to operator errors and inefficient handling of physical queues for problem setup and breakdown by integrating the card-to-tape, compute, and tape-to-printer functions. This is often referred to as automating the machine room.
- Reduction of inter-job computational processor idle time by dynamic scheduling of jobs for the main processor in order to

overlap tape mounting and demounting with processing of other jobs.

- Reduction of job breakdown time by dynamic scheduling of output printers and punch according to priority, printer paper conditions, etc.
- Reduction in the processing time of individual jobs by sharing the functions of an input output control system between the two processors in parallel and by more buffering on both input and output.
- Expansion of usable program storage space in the computational processor by removing from it much of the IBJOB MONITOR and INPUT OUTPUT CONTROL SYSTEM.
- Reduction of computational processor idle time due to the printing on line of accounting information and operator instructions by transferral to the monitor processor.
- Reduction of computational processor idle time due to erroneous or undesired halts by positive control by the monitor processor to initiate dump and restart procedures.
- Reduction in excess file access time due to random addressing by control and scheduling of all file references by the monitor processor.

Furthermore, variations of the directly coupled system described may attack:

- Extension of system applicability by the relatively easy accommodation of remotely located terminals by the monitor processor.
- Reduction of fruitless attempts by the computational processor to assemble or compile programs which result in detection of rather trivial program errors by a degree of pre-editing by the monitor processor.
- Reduction of computational processor initial job loading time by the execution of some preparatory functions by the monitor processor in advance of load time.

The cost of the basic system is in the same area as, and possibly cheaper than, a 7090 supported by two off-line 1401's. It is clear that the system is practical and offers many advantages over today's mode of operation. Required reprogramming, however, is far from trivial.

The day is long past when computers can be rated in terms of arithmetic processing speed. They must be rated only in terms of how they are used. Multiprocessing concepts promise complex but advantageous use of equipment to better serve the ultimate user.

ACKNOWLEDGEMENT

The author is indebted to Barbara Broome, D. E. Freeman and R. E. Moore, of the IBM DATA PROCESSING DIVISION for much

analytical and simulation work. Considerable contribution has also been made by the IBM DATA SYSTEMS DIVISION, particularly by M. E. Drummond and J. F. Dillon and their staffs.

FOOTNOTES AND CITED REFERENCES

1. These and other specific programming systems mentioned within the paper are described in References 2 through 6.
2. A. S. Noble, Jr., "Design of an Integrated Programming and Operating System—Part I: System Considerations and the Monitor," *IBM Systems Journal*, **2**, 153, June 1963.
3. R. B. Talmadge, "Design of an Integrated Programming and Operating System—Part II: The Assembly Program and Its Language," *IBM Systems Journal*, **2**, 162, June 1963.
4. R. Hedburg, "Design of an Integrated Programming and Operating System —Part III: The Expanded Function of the Loader," *IBM Systems Journal*, **2**, 298, this issue.
5. R. Larner, "Design of an Integrated Programming and Operating System —Part IV: The System's FORTRAN Compiler," *IBM Systems Journal*, **2**, 311, this issue.
6. R. T. Dorrance, "Design of an Integrated Programming and Operating System—Part V: The System's COBOL Compiler," *IBM Systems Journal*, **2**, 322, this issue.
7. F. R. Baldwin, W. B. Gibson and C. B. Poland, "A Multiprocessing Approach to a Large Computer System," *IBM Systems Journal*, **1**, 64, September, 1962.