This paper is a survey of changes to virtual machine interfaces, implementation, architecture, and simulation techniques as they affect IBM System/370 and 303X (3031, 3032, 3033) processors, the system control program to which virtual machines interface, and other virtual machines executing on the same real computing system or elsewhere. The paper seeks to summarize such changes and provide a perspective on the virtual machine environment. New uses of virtual machine subsystems are discussed as they relate to inter-virtual-machine communication.

The changing virtual machine environment: Interfaces to real hardware, virtual hardware, and other virtual machines

by R. A. MacKinnon

When IBM introduced virtual machine products with CP-67 on the System/360 Model 67, an early view of the uniqueness of virtual machines focused on the isolation of one virtual machine from another.[1,2] CP-67 was able to provide System/360 hardware systems with a variety of operating system environments, or virtual machines, all independent of each other. This was accomplished with dynamic address translation hardware and the use of multiple virtual address spaces, hypervision by the control program (CP), and CP's handling of the real machine. System/360 split instructions into privileged system control instructions and nonprivileged computational instructions, enabling CP to achieve its objectives with virtual machines by exploiting the real hardware whenever possible.

CP-67's ability to handle a variety of virtual machine software environments and the separation and isolation of virtual machines made the system attractive to users. Development of the Cambridge Monitor System (CMS) provided conversational computing to CP-67, which became known as CP-67/CMS.[3]

Since 1972, the use of virtual machines has become more widespread with the availability of IBM's Virtual Machine Facility/370 (VM/370) for many models of System/370.[4] Virtual machine archi-

tecture and the implementation of VM/370 are discussed in References 5, 6, and 7. This paper examines how VM/370 has progressed beyond its predecessor, CP-67/CMS.

I. Outline of the paper

Various hardware and software implementations have been designed to improve VM/370 system performance, as well as the performance of individual virtual machines and CP. Initially, a System/370 hardware implementation called *virtual machine assist* was designed to enhance the execution of privileged instructions and supervisor calls (normally associated with virtual machine operating systems). VM assist is discussed in Section II.

VM assist extended hardware capability beyond the defined System/370 instruction set while preserving the virtual machine's view of *its* instruction set. Certain processors are now provided with information about the virtual machine execution environment, and these processors are optimized or tailored to enhance VM/370 performance. On certain processors, this extension has continued to enhance CP execution and expand VM assist through more hardware support. Again, the objective is to improve system execution time by assisting, with direct hardware execution, routines that handle certain CP functions. This further development (which is additional to VM assist) is called *Extended Control Program Support:VM/370 (ECPS:VM/370)*. It is covered in Section III.

CMS is discussed by Seawright and MacKinnon in the preceding paper.[3] Other major trends in virtual machine interfaces and execution can be found in some non-CMS environments. *Handshaking* is the name used here for the changes to the DOS/VS and OS/VS1 system control programs that enable them to recognize and take advantage of their execution in a virtual machine environment.[8] While the implementation is different in each of these operating systems, their handshaking objectives remain the same—bypassing, eliminating, or reducing functions that are redundant or operationally inefficient in the virtual machine environment. The result has been enhanced performance of DOS/VS and OS/VS1 in virtual machines and improved operation of virtual machine systems. Handshaking is discussed in Section IV.

However, while handshaking complements the VM assist and ECPS:VM/370 hardware functions, it does not employ hardware. Even with handshaking, the problem programs managed by OS/VS1 and DOS/VS do not recognize their virtual machine environment. This holds true for CMS and Multiple Virtual Storage (MVS) problem state programs as well. They regard their operating system as controlling a real machine.

Section V discusses the use of virtual machine architecture for system development and implementation. Emphasis is on inter-virtual-machine communication, which departs from the earlier view that total isolation was to many users a dominant view or desirable.

Section VI treats control program assist approaches in environments other than VM. New instructions in System/370 Models 158-3 and 168-3 and in the 303X processors for the MVS System Extensions Program Product (MVS/SEPP)[9,10] are discussed both as examples of how control program assists extend beyond VM/370 and as they relate to MVS in the virtual machine environment. These assists apply to MVS/SEPP even when executed in a virtual machine. The MVS discussion includes CP changes designed to achieve "partial" (one-sided) handshaking on behalf of MVS. This section concludes with a discussion of Extended Control Program Support (ECPS) for OS/VS1 and of the APL assist for microcode execution of APL statements, to amplify the point that hardware and software changes in VM/370 also appear elsewhere.

Finally, possible future evolutionary trends are charted in the Conclusions.

II. Virtual machine assist

VM assist is a hardware implementation designed to improve the performance of some VM/370 virtual machines, and thereby enhance the performance of that particular system.[11-13] This discussion concentrates on the function of VM assist and how it helps the execution of virtual machines, rather than its actual implementation on various CPU's. The primary interface between virtual machines and CP occurs when the operating system in the virtual machine executes a privileged instruction or a supervisor call (SVC). As an additional interface, CMS employs the DIAGNOSE instruction for functions such as disk I/O and depends on CP for its operation. CMS can execute only in a virtual machine. The interfaces between virtual machines and CP are discussed by Seawright and MacKinnon.[3]

So what VM assist seeks to do is emulate, whenever possible, certain CP routines that simulate the instruction that the virtual machine executed. VM assist produces results that are functionally equivalent to CP's results. Significantly, VM assist executes directly, without CP, and thus can dispense with the interruption handling and redispatching associated with the transfer of control between virtual machines and CP. There are exceptions to this support. The non-high-performance and not-well-definable virtual machine uses of certain privileged instructions are not assisted.

Figure 1 Control register 6

Bit	*Function and use*
0	Virtual machine assists ON/OFF—used by VM assist, EVMA, and VITA; can be turned on and off for each virtual machine.
1	Virtual machine problem or supervisor state—set by CP and VM assist; will determine whether an instruction was issued in privileged state.
2	ISK and SSK instructions—whether VM assist should handle these instructions.
3	System/360 or System/370 mode—helps VM assist and ECPS:VM/370 determine which instruction set is valid for the virtual machine.
4	SVC handling—set by CP to tell VM assist whether non-SVC 76's should be reflected to the virtual machine.
5	Shadow table fixup—activates handling by VM assist.
6	CP assist—set by CP for entire system; in conjunction with bit 0, defines which assists (VM assist, EVMA, VITA, CPA) are active.
7	Virtual interval timer assist (VITA) for ECPS:VM/370
8-28	Real address of VM pointer list.
29	Emulation of System/370 extended instructions executed in virtual machine by virtual machine extended-facility assist.
30-31	Reserved and undefined.

VM assist relies on control register 6[14] for key information to govern its actions for individual virtual machines. CP manipulates its settings of control register 6 as part of dispatching the virtual machine or establishing its contents for CP execution. Figure 1 maps this register. As an example, note that VM assist handles privileged instructions only when it is ON (bit 0), the CPU is in the real problem state (from the current program status word), and the virtual machine is in the virtual supervisor state (bit 1).

privileged operation execution

The specific instructions handled by VM assist are listed in Table 1.[15,16] A distinction is made between System/370-only and System/370 and System/360 virtual machines. That distinction is indicated to VM assist by bit 3 in control register 6.

The System/370-exclusive instructions are most likely to be found in operating systems that support virtual storage management. Early System/360 operating systems such as DOS and OS do not use them at all and thus derive less benefit from VM assist.

Table 1 Privileged instructions handled by VM assist

System/360 and System/370	*System/370 only*
INSERT STORAGE KEY (ISK)	INSERT PSW KEY (IPK)
LOAD PSW (LPSW)	LOAD REAL ADDRESS (LRA)
SET STORAGE KEY (SSK)	RESET REFERENCE BIT (RRB)
SET SYSTEM MASK (SSM)	SET PSW KEY FROM ADDRESS (SPKA)
	STORE CONTROL (STCTL)
	STORE THEN AND SYSTEM MASK (STNSM)
	STORE THEN OR SYSTEM MASK (STOSM)

As VM assist encounters one of the privileged instructions listed in Table 1, it performs the appropriate emulation and then returns control to that virtual machine's next instruction. This process bypasses the following CP routines: interruption handling, analysis of the operation to be simulated, the actual simulation, and invocation of CP's dispatch routine. Performance improvement comes from hardware implementation rather than software execution, and from having to perform only the actual simulation.

SVC handling

When executing an SVC instruction, a virtual machine does not have to be in the virtual supervisor state. Essentially, SVC's are handled exactly as outlined for the VM-assist-related privileged instructions. The exception is SVC 76 which has been reserved arbitrarily for CP to handle or reflect real machine error information to the virtual machine.[17] Thus VM assist does not handle SVC 76.

shadow page table management

Virtual machines operating in extended control mode with the dynamic address translation (DAT) facility ON require that special page and segment tables be created and maintained by CP.[7,12,18] These "shadow" tables enable the virtual machine operating system to utilize DAT hardware and manage its virtual storage as it would on a real machine. CP manages the "real" storage associated with the virtual machines through demand paging and its own set of segment and page tables. The shadow tables enable CP to let the virtual machine manage its virtual storage while CP manages its real storage. Note that the virtual machine's page size may differ from CP's page size.

VM assist's role in shadow table management is to receive translation exception conditions caused by the virtual machine and handle them without causing an actual translation interruption. Specifically, instead of directing the translation exception to CP, VM assist checks to see if the page that caused the translation exception is actually in real storage. This is determined by check-

ing the virtual operating system's DAT tables and the real segment and page tables. VM assist finds both sets of tables through the virtual machine pointer list addressed via control register 6.[19] If VM assist finds that the desired page is in fact in real storage, it marks the shadow page table entry valid, places the proper page frame address within the entry, and keeps control in the virtual machine. If the desired page is not in real storage, VM assist reflects a translation exception to CP. In summary, this aspect of VM assist reduces the occasion for an actual interruption and CP analysis when the needed page is already available.

selective use of VM assist

Since not all models of System/370 provide VM assist, CP's initial program load (IPL) sequence determines VM assist's availability and activates it when present, using bit settings in control register 6 (see Figure 1). The SET command provides the CP system operator with a means to deactivate then reactivate the VM assist facility through this register. Individual virtual machine console operators can disable or enable VM assist services for their specific virtual machines through the SET command. Further selectivity can be exercised by disabling the SVC handling component of VM assist with the SET command.

The result is maximum flexibility for the installation and the individual user during evaluation of VM assist, performance analysis, and benchmarking. The effect of VM assist on the overall system, and for individual virtual machine execution, can be assessed without any CP changes.

benefits of VM assist

VM assist, then, is a first hardware step toward partitioning CP simulation services between CP and VM assist hardware facilities. The time used by VM assist is problem state time and is so reflected. In most cases, VM assist uses a significantly reduced amount of real supervisor state time and a slightly increased amount of real problem state time. The effects of this change on overall system throughput, response time, and specific virtual machine environments are shown in Table 2.

The information in Table 2 is taken from Horton, Wagler, and Tallman.[13] The table summarizes results of a variety of measurements intended to demonstrate approximations of the effects of VM assist. The results do not necessarily represent typical operating environments. The tests, in fact, were run using VM/370 Release 3, Program Level Change 2. The important considerations are the relative improvements, rather than precision or relevance to current releases or maintenance levels. In no way is this series of benchmarks intended to provide a comparison among various processors or system control programs.

Across a spectrum of operating systems (DOS/VS, OS/VS1, OS/VS2 SVS) issuing privileged instructions and SVC's, the benchmark re-

Table 2 Effects of VM assist on system throughput and specific virtual machines.

	Model 135		*Model 145*		*Model 158*	
	DOS/VS	*VS1*	*DOS/VS*	*VS1*	*VS1*	*VS2*
Elapsed time (seconds)						
Native	2788	3035	2150	1418	1386	572
Virtual machine	8172	11 598	4520	4089	3769	2696
Virtual machine with VM assist	4226	4063	2723	2024	2004	1149
Relative batch throughput without VM assist	0.34	0.26	0.48	0.35	0.37	0.21
Relative batch throughput with VM assist	0.66	0.75	0.79	0.70	0.69	0.50
Reduction in supervisor state time (Vm assist vs. non-VM assist)	74%	89%	73%	86%	82%	69%
Reduction in elapsed time (VM assist vs. non-VM assist)	48%	65%	40%	51%	47%	57%
Reduction in total number of privileged instructions simulated by VM/370 (VM assist vs. non-VM assist)	87%	95%	86%	94%	91%	74%

sults portray the effects of VM assist in terms of native throughput, virtual machine elapsed time, relative batch throughput,[20] supervisor state and elapsed time, and the number of privileged operations simulated by CP. The tests were run using System/370 Models 135, 145, and 158.

Although many virtual machine environments benefit from VM assist, clearly not all do. For example, CMS virtual machines interface to CP for specific services and thus do not execute privileged instructions as frequently as non-CMS virtual machines. The DIAGNOSE instruction is CMS's primary interface to CP. Also, non-virtual operating systems such as DOS/360, OS/MFT, and OS/MVT do not benefit as directly from VM assist because they are less apt to issue the range of instructions aided by VM assist—those associated with extended control mode and DAT operation on System/370.

Thus the implementation of VM assist was a starting point for addressing virtual machine performance problems. How this was done has been discussed. The next section shows how this point of departure has broadened considerably for VM/370 on certain System/370 configurations.

III. Extended Control Program Support

Extended Control Program Support:VM/370 (ECPS:VM)[7,21] provides for the further utilization of hardware to enhance VM/370 performance on Models 135-3, 138, 145-3, and 148 of System/370. ECPS:VM works in conjunction with VM assist, providing even more comprehensive services for the virtual machines and ex-

tending the assist concept to CP execution. ECPS is controlled by bit settings in control register 6 (see Figure 1) as well as by new System/370 instructions which change the hardware interface between CP and the CPU. As discussed below, this enhancement and extension consists of new functions completely handled by hardware, functions that are partly handled by hardware, and functions that are a combination. ECPS:VM has an expanded VM assist component, a virtual interval timer assist component, and a control program assist component, and it works in conjunction with VM assist.

VM assist

ECPS:VM works in conjunction with VM assist but does not include it. Thus when VM assist handles certain instructions—LOAD PSW (LPSW), SET SYSTEM MASK (SSM), STORE THEN AND SYSTEM MASK (STNSM), STORE THEN OR SYSTEM MASK (STOSM)—it calls on the expanded VM assist (EVMA) component of ECPS to complete the simulation. EVMA undertakes the simulation when entered from VM assist. Should it not be able to complete the operation, it causes CP to simulate the instruction by directly passing control to CP. CP then handles to completion as if VM assist were not originally available. Figure 2 shows this sequence, using LPSW as an example.

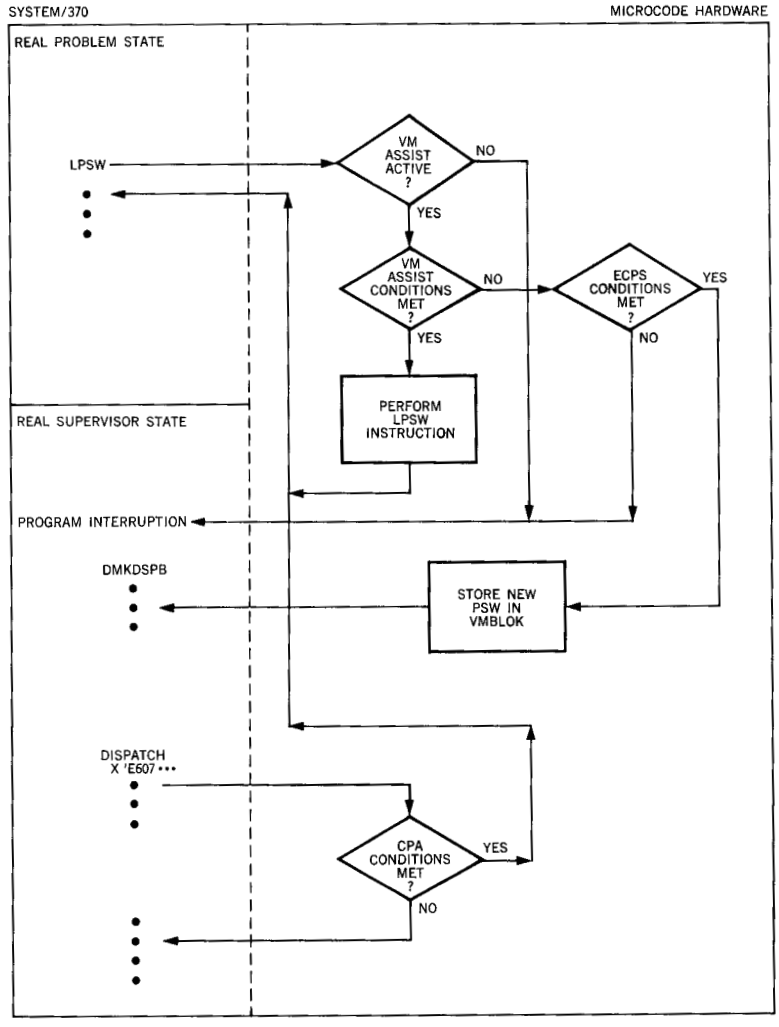
When VM assist or EVMA can handle one of the designated privileged instructions, control returns to the virtual machine without recourse to CP. Supervisor calls are handled as discussed in Section II, above.

expanded VM assist

Along with the partial handling discussed above for VM assist, EVMA provides complete handling of the privileged instructions PURGE TRANSLATION LOOKASIDE BUFFER (PTLB), STORE CPU TIMER (STPT), and TEST CHANNEL (TCH). In addition, portions of other new functions for SET CLOCK COMPARATOR (SCKC), START I/O (SIO), START I/O FAST RELEASE (SIOF), and SET CPU TIMER (SPT) are handled by EVMA, with CP simulation completing the functions.

Insight into how EVMA and CP work together can be gained by examining the execution of DIAGNOSE. Starting with the VM/370 System Extensions Program Product (SEPP) and the Basic System Extensions Program Product (BSEPP),[22] software now supports EVMA hardware for assisting the DIAGNOSE instruction when executed by a virtual machine. When issued under assisted conditions, EVMA bypasses the program exception interruption and CP's first-level interruption handler and transfers control directly to CP routine DMKHVC, where DIAGNOSE code analysis and simulation actually take place. Section V covers the use of DIAGNOSE by CMS and other virtual machines, but it is sigificant to point out here that this major interface is now included among the assists provided by hardware.

Figure 2 Interaction of software and microcode hardware (EVMA and CPA) for VM assist and ECPS:VM/370



The degree of instruction simulation performed by EVMA in cases of partial execution differs by instruction. At the very least, EVMA hardware prepares certain "housekeeping" functions, such as register saving and unloading and decoding of the privileged instruction. Other possibilities include partial simulation or no simulation, leaving CP to provide such function. For more information on this topic, see Reference 21.

Whether or not VM assist or EVMA calls on CP, control eventually returns to the virtual machine, which does not recognize whether hypervision is performed by CP, hardware, or both.

virtual interval timer assist

The virtual interval timer assist (VITA) component of ECPS:VM maintains the virtual interval timer (in location 80 of page 0 of the

virtual machine) and handles interruptions related to the virtual timer. VITA explicitly has the following responsibilities:

- When a virtual machine is executing, VITA decrements its virtual interval timer whenever the real interval timer decrements.
- If page 0 of the virtual machine is not in real storage at this time (CP may have paged-out this page), VITA maintains the timer in this virtual machine's central control block (VMBLOK).
- As the virtual interval timer turns to a negative value, VITA seeks to present a timer interruption to the virtual machine if possible. Examples in which this is not possible are timer interruptions disabled in the virtual machine and page 0 not in real storage. If the virtual machine cannot accept such an interruption, VITA presents a virtual interval-timer interruption to CP in such a way that CP can differentiate between real and virtual interval-timer interruptions. CP then reflects the interruption back to the virtual machine after handling the situation that prevented VITA from doing it directly. In short, the interruption is stacked by CP instead of by the hardware.

VITA hardware function benefits virtual machines by eliminating programming routines and enhances accuracy in timer servicing because necessary interruptions can be presented faster.

control program assist

VM assist, EVMA, and VITA all assist a specific virtual machine. New System/370 instructions, which provide for assisting CP, can be generally beneficial to VM/370 and reduce more general "overhead." Starting with VM/370 Version 3, Program Level Change 8, programming support is provided for those models of System/370 that have the ECPS:VM facility installed. The control program assist (CPA) component utilizes new System/370 instructions to assist certain CP routines. The presence of this capability is determined as CP executes a new privileged instruction, STORE ECPS/VM LEVEL IDENTIFIER (STECPSVM), which detects whether ECPS:VM is installed and operating at the appropriate level. If not, the new System/370 instructions for ECPS are made *no-operations*, and CP uses existing software routines rather than ECPS. Next, CP determines whether VM assist is installed and, if so, activates it. Thus VM/370 can support machines with ECPS, with only VM assist, or with neither, as determined at CP's IPL time.

The new System/370 instructions for CP's use have the extended storage-to-storage format shown in Figure 3, in which X'E6' is the operation code of the CPA instruction and X'cc' defines the specific function to be performed. The two operands provide parameters to the specific assist function. These instructions are not defined in the *System/370 Principles of Operation*, nor do they have assembler-language mnemonics. They appear as DC statements in the source code distributed for CP. The specific CP functions assisted by the CPA instructions are listed in the Appendix.

Figure 3 Extended storage-to-storage format of new System/370 instructions for CP's use

Bits 0	8	16	20	32	36
OP CODE X'E6'	Extended OP CODE X'cc'	B1	D1	B2	D2

component activation

ECPS:VM is implemented with considerable flexibility as to which components support VM/370 at any given moment. During IPL of CP, all components are activated, but the system operator can disable then enable all ECPS components, or disable then enable them selectively. At the virtual machine level, the console operator for the virtual machine can disable then enable EVMA and VITA. The virtual machine cannot affect the status of the CPA function. With VM BSEPP and SEPP software, additional selectivity is supported for EVMA. Through an assist control field in the VM list addressed by control register 6, selected instructions can be enabled or disabled for EVMA support.[23] Such flexibility is helpful both in maintaining system availability should hardware problems arise with specific ECPS hardware modules and in achieving a high degree of system portability across CPU configurations without CP change.

ECPS:VM significantly extends System/370 support of VM/370 execution. It builds on the capabilities introduced by VM assist and extends the assist philosophy to CP as well. In sum, hardware can assist the virtual machine environment, CP can benefit from assist hardware, and CPU control can pass from virtual machine privileged operation to VM assist or EVMA hardware to CP (for completion) and back to the virtual machine, as in Figure 2.

The consequence is improved performance of certain virtual machine operations and certain CP routines. The extent to which any given system benefits is a function of understanding ECPS functions and program behavior. CMS does not benefit as much from VM assist as do virtual machines that run DOS/VS or OS/VS. CMS does benefit, however, from DIAGNOSE, which is a primary interface to CP, and from CPA operation once control has passed from CMS to CP. Thus the mix of CMS and other virtual machines determines the benefits provided by ECPS/VM (as distinct from VM assist).

Table 3 Effect of VM assist on CMS batch processing on System/370 Model 145

	Percentage of time in indicated state		*Paging rate* (per second)	*Real problem percentage**
	Real supervisor	*Real problem*		
Without VM assist	60.3	39.7	39.8	30.6
With VM assist	57.9	42.1	42.8	29.4
Difference	− 3.98	+ 6.05	+ 7.54	− 3.92

*Percentage of total real problem state time used by virtual problem state

Table 4 Effect of VM assist and ECPS:VM on CMS batch processing on System/370 Model 148 configured like Model 145

	Percentage of time in indicated state		*Paging rate* (per second)	*Real problem percentage**
	Real supervisor	*Real problem*		
VM assist only	46.5	52.7	45.6	26.0
Full ECPS	33.6	62.8	47.0	25.6
Difference	−27.7	+19.2	+ 3.1	− 1.5

*Percentage of total real problem state time used by virtual problem state

As noted in the earlier discussion of VM assist, ECPS has further addressed VM performance problems and made even greater contributions to such improvements.

Tables 3 and 4 summarize the results of benchmarks that provide approximate comparisons of the assists discussed above. CMS virtual machines issue relatively few privileged instructions or supervisor calls, so it is interesting to consider how VM assist affects CMS environments. Table 3 shows how VM assist can affect a CMS batch processing application on a System/370 Model 145. The same job stream was run on a Model 148 configured like the Model 145. The results, summarized in Table 4, provide insight into the effects of VM assist and ECPS/VM during CMS execution.

IV. Handshaking

A common view of the virtual machine environment stresses the isolation of the virtual machine from the real environment. That is, a program (including operating system code) that runs in a virtual machine does not recognize that CP controls the execution of privileged instructions, handles real interruptions and other

asynchronous events, and reflects events back to the virtual machine programming exactly as if the virtual machine were a real machine (with the exception of timing considerations). This capability allows diverse System/360 and System/370 programs to be multiprogrammed without interfering with or causing problems for one another. CP addresses discrepancies between the virtual and real environments.

Handshaking changes the above view for DOS/VS and OS/VS1 in that they are given the information that they are executing in virtual machines and can take certain actions using that information. Virtual machine isolation remains; what changes is the behavior of the operating system within the virtual machine and the interface to CP. It is important to understand that in these operating systems, the problem program partition (or user) does not recognize the virtual machine environment, and the interface to its operating system remains unchanged.

In simulating multiple virtual machines, CP's role is to provide services to those virtual machines (as requested) and handle the real hardware system. As to what is actually going on *within* a virtual machine, CP has very limited information. CP is largely restricted to reflecting back to the virtual machine's operating system conditions that relate to it. For example, CP is not involved in how an operating system manages multiprogramming and multitasking within a virtual machine. They are the responsibility of the virtual machine's operating system. CP gives control to a virtual machine, then it is up to the virtual machine's operating system to dispatch units of work according to its own priority scheme.

CP regards virtual machine execution as a continuum. The initiation and termination of jobs and tasks within the virtual machine are hidden. CP only regards the virtual machine as a whole. This has proved a mixed blessing, for it has both ensured the transparency of CP to the virtual machine and distorted the time spectrum between a virtual machine's partition or task losing control and regaining control from its operating system. Given this distortion, the relative batch throughput or transaction rate of a non-CMS virtual machine, or a VM system as a whole, can be adversely affected. Handshaking directly addresses these functional problems and provides some operational improvements.

OS/VS1 handshaking

Analysis of OS/VS1 execution as a virtual machine quickly leads to identification of several areas of duplication in VS1 and CP, where different implementations or operational improvements could enhance the VS1 virtual machine environment.[7,24] Of primary importance is to provide VS1 with the information that it is interfacing to

CP rather than to a real machine. A system generation option gives VS1 new capability, so that at IPL time it receives that information. During IPL, it issues a STORE CPU ID instruction to discover whether it is in control of a real machine. If not, it issues a DIAGNOSE instruction to ensure that CP will provide handshaking support. Then VS1 goes through the nucleus initialization process to activate handshaking on its behalf.

Thus handshaking is two-sided, with cooperation on both sides of the interface between CP and the virtual machine. Implementations that have resulted from the above process are nonpaged mode, pseudo page-fault handling, CP spool files, and I/O-related items. They are discussed in the following paragraphs.

nonpaged mode

Provided enough virtual machine storage is defined, VS1 marks all virtual page frames *fixed*, builds page tables only in the systems queue area, and disables demand paging. It does not open an external page storage file or attempt to translate the channel programs it uses, and it reduces its use of the LOAD REAL ADDRESS and INSERT STORAGE KEY instructions (keys are handled by a table rather than real hardware). Here OS/VS1 is turning the business of demand paging over to CP. While double paging is thus eliminated as a programming overhead item, the virtual machine executes with DAT hardware ON.

pseudo page-fault handling

Through the SET PAGEX command, the virtual machine console operator can exercise an option that affects how CP handles real page faults attributed to an OS/VS1 virtual machine. When a page fault occurs, CP gains control. The key question is which virtual machine should be dispatched by CP while the page I/O operation is under way? Ordinarily, CP places the entire virtual machine in a *page wait* status and dispatches another virtual machine. When exercised, this pseudo page-fault facility causes CP to reflect a special page fault to VS1 (even though the latter is not paging). VS1 makes specific use of this situation by marking the affected partition or task as being in a *page wait* status; then it is free to dispatch another partition or task. As a result, VS1 can multiprogram properly.

When CP completes the real paging operation, it reflects page I/O completion to VS1 to clear *page wait* for that partition or task. This accommodation by CP enhances the performance of multiprogramming or multitasking within the VS1 virtual machine. When the multiprogramming level of the virtual machine is low, with no multitasking, pseudo page-fault handling can be disabled by the VS1 console operator, and CP then resumes normal dispatching of virtual machines.

Looking at the system as a whole, the installation might utilize PAGEX OFF to skew dispatching emphasis to a partition that is

executing telecommunications and a program like the Customer Information Control System (CICS), which does its own multitasking.

CP spool files

Since CP does not detect the end of a job or job step within a VS1 virtual machine, operational problems arise as to when CP is to start processing spool file output (punch or printer). Without some mechanism, a manual step is needed: an operator must close the VS1 spool file to release the output to CP. Handshaking provides a VS1 interface to CP for this explicit purpose. A DIAGNOSE instruction issued by VS1 signals CP that the job or job step has been completed and that CP's spool output operations can be scheduled. Here is a good example of an operational improvement resulting from handshaking techniques.

I/O-related items

In any operating system that supports demand paging, the I/O supervisor increases system overhead. This is especially true of code that requires translation of channel programs. Handshaking improves the way VS1 handles I/O in such cases. VS1 neither translates channel programs nor builds indirect data addressing lists when handshaking is operative. Similarly, VS1 allows CP to handle IBM 2314 and 2319 direct-access-storage seek separation operations, and it refrains from issuing a TEST CHANNEL (TCH) instruction prior to executing the START I/O instruction. These functions are performed later by CP.

Particularly thorny for virtual machine I/O is the modification of channel programs after they start. Normally CP does not guarantee proper handling of such channel programs, but the VS1 Basic Telecommunications Access Method (BTAM) is an exception when its *autopoll* feature is in use. Autopoll modifies the virtual communications channel programs for a line. Without handshaking, CP utilizes flags in the real channel program to signal it to inspect the BTAM virtual channel program for changes. Handshaking eliminates this approach by providing a VS1 DIAGNOSE instruction, which signals CP at the time of change to allow CP to update the real channel program it builds and maintains. CP's continuous investigation of BTAM is thereby eliminated.

A somewhat related implementation applies to the Telecommunications Access Method (TCAM) at Release 5. Regardless of whether its operating system supports handshaking, TCAM can be generated for execution in a virtual machine, with DIAGNOSE instructions to signal CP whenever a TCAM channel program is being modified. The objective is to allow such programs to run in a virtual machine as paged rather than with the earlier nonpaging restriction. This process can be considered handshaking for a special purpose.

DOS/VS handshaking

Handshaking implementations for DOS/VS[25,26] differ considerably from those for OS/VS1, but the objectives remain identical. The main difference involves virtual-timer updating, which enables the accounting routines of DOS/VS to more accurately reflect timer settings when a DOS/VS job terminates. DOS/VS signals CP so that it can update the virtual interval timer. CP can also be signaled when DOS/VS changes the timer's value. This is a programming approach to what the VITA hardware component of ECPS provides on System/370 Models 135-3, 145-3, 138, and 148.

Handshaking is a system generation option for DOS/VS but, unlike VS1, the resulting system can be executed only as a virtual machine. It is not an IPL option.

Performance enhancement

Handshaking is a programming approach to enhancing the performance and operational effectiveness of virtual machines. It can operate with the hardware assists discussed above. Alone, handshaking can both aid virtual machine execution and reduce real supervisor state (CP) execution time by reducing the number of privileged instructions, virtual machine paging I/O, etc. To a large degree, assists and handshaking try to solve the same problems, so knowledge of where specific benefits accrue requires considerable knowledge of the behavior of virtual machines. For instance, elimination of double paging greatly reduces the I/O activity that CP must handle and the number of privileged instructions executed in the virtual machine. It also ensures that when a virtual machine has control, execution will tend to be more in the virtual problem state than in the virtual supervisor state; more work is being done by executing application user code directly.

The pseudo page-fault facility is a good example of how CP makes use of the information that multiprogramming or multitasking is going on in a virtual machine. SPOOL CLOSE addresses the problem of virtual machine job or task transition by allowing the operating system in the virtual machine to notify CP by means of the DIAGNOSE instruction. The DIAGNOSE interface, then, has found use beyond CMS, and its increasing usefulness benefits from ECPS:VM's assistance.

results of a benchmark

The performance improvement that can result from handshaking is indicated by results of a benchmark conducted at the IBM World Trade Systems Center in Poughkeepsie, New York. The benchmark consisted of running 12 jobs on a System/370 Model 158 with VM assist and prototype handshaking code. Ten jobs executed the COMPILE LINK AND GO (CLG) step in FORTRAN. One COBOL job executed five CLG steps with SORT called by COBOL,

and one job, also in COBOL, executed CLG, SORT, and CLG steps. Operator setup was minimal, and native CPU utilization averaged about 90 percent. The following results are pertinent:

- *Relative batch throughput:* With handshaking, when CP paged an OS/VS1 virtual machine, the relative batch throughput increased 35 percent (from 0.55 to 0.74) while total CPU time decreased 29 percent (from 781 to 557 seconds). When the benchmark was rerun with the OS/VS1 virtual machine executing out of VM's virtual-equals-real area, handshaking improved relative batch throughput by 14 percent (from 0.69 to 0.79), and CPU time decreased by 20 percent (from 717 to 570 seconds).
- *Pseudo page-fault handling:* The benchmark used six initiators, and, with PAGEX OFF, nine of the 12 jobs ran in VS1 partitions 0–2, while partitions 3–5 did not get beyond a single job. With PAGEX ON, partitions 3 and 4 each ran a second job. Thus, although pseudo page-fault handling did not affect performance throughout the benchmark, it did affect the level of multiprogramming by OS/VS1.
- *Effect of handshaking apart from VM assist:* Since the runs described above used VM assist, two runs were made *without* VM assist—one with handshaking and PAGEX OFF, and one with no handshaking. Relative batch throughput was 0.54 with handshaking and 0.24 without.

V. Inter-virtual-machine communication

Emphasis to this point has been on hardware and programming approaches to improving the performance and function of VM/370 and specific virtual machines. As indicated by Seawright and MacKinnon,[3] this improvement was an enabling event which resulted in increased acceptance of VM/370 for many CPU configurations, with a diversity of end-user applications.

The balance of this paper seeks to review and assess another trend which has similarly affected VM/370. Inter-virtual-machine communication, in which the objective is to send data or control information between virtual machines, is examined. Mechanisms for accomplishing this objective are described, and motivations behind this trend are examined. Examples show how inter-virtual-machine communication has increased the potential role of virtual machines.

It is important to note the growth of virtual machines as subsystems. Users have conducted experiments and designed systems to create virtual-machine-resident software with characteristics different from those discussed above. They are subsystems

that have information about and are dependent on CP and the virtual machine environment. In this sense, they more closely resemble CMS than other operating systems.

Remote Spooling Communications Subsystem

A subsystem incorporated by IBM into VM/370 was the Remote Spooling Communications Subsystem (RSCS),[27] which operates as a separate virtual machine. RSCS manages spool files transmitted between virtual machines and remote-job-entry stations, between remote CPU's operating as remote-job-entry stations, and between other CPU's operating HASP or ASP spooling components and RSCS (and thereby viewing VM/370 as a remote work station). RSCS also manages files sent from remote work stations and CPU's and destined either for machines within this VM/RSCS system or for output on other work stations or CPU's connected to this system.

RSCS is a special-purpose operating system. It contains its own multitasking supervisor, and it provides storage and task management, line drivers for the communications links, and service routines for command processing. It can operate only in a virtual machine environment. It can operate in disconnected mode if communication with the RSCS virtual machine operator is not required.

RSCS interfaces to CP via the latter's local spool files and by use of a DIAGNOSE interface. It is both a special-purpose subsystem constructed as a virtual machine, and a means by which a virtual machine can transmit data outside itself (and, indeed, outside the real machine as well). RSCS was an early step toward a virtual machine networking capability.

CP's local spooling capability is mentioned only in the context of the use of spool files by RSCS for sending and receiving data externally on communications lines. Spool files also have been used by virtual machines to exchange data within the same real machine, in that one virtual machine can send data to its card punch or printer for spooling to the card reader of another virtual machine. This approach has the advantage of using a well known and externally defined function of CP while maintaining the isolation and logical view of virtual machines.

A disadvantage for transaction handling lies primarily in the inflexibility of the data formats (unit record) and the I/O overhead associated with transcribing the data to and from the spool packs. What evolved was a series of techniques for efficient inter-virtual-machine communication employing storage-to-storage data transfer. This spurred development of additional subsystems based on virtual machine architecture.

An early approach was use of the virtual channel-to-channel adapter (CTCA) support already in CP. Originally it was intended to facilitate testing of ASP loosely coupled multiprocessing configurations in a virtual machine environment. Through a CP COUPLE command, two virtual machines are connected through a virtual CTCA path, and the READ/WRITE I/O commands issued by these machines are simulated by CP. Data can then be exchanged between virtual address spaces via CP's *move* instruction rather than by an I/O operation (assuming that no page fault occurs).

other approaches—SPY and VMCF

A more elaborate approach was devised by A. N. Chandra at IBM's Thomas J. Watson Research Center.[28] The resulting virtual machine, called SPY, was particularly useful because it managed a variety of special-purpose virtual machines (for data management or networking, for instance), it recorded virtual machine accesses and linkages, and—most important for this discussion—it provided another protocol and facility for the interchange of data between virtual machines. CP's Virtual Machine Communications Facility (VMCF) is based on this part of SPY.[7,29,30]

VMCF uses two principal interfaces to allow virtual machines to communicate. First, a DIAGNOSE instruction requests special VMCF facilities from CP. The second interface is an external interruption which serves as a signal for notification and synchronization of transmissions and acknowledgments between virtual machines. CP generates these interruptions for both sending and receiving virtual machines.

VMCF provides for transfer between storage—the virtual address space of a sending virtual machine (called the source) and receiving virtual machines (called sinks). A single source is able to send to more than one sink. In the process, two real page frames are locked. VMCF, then, formalizes procedures for inter-virtual-machine communication and recognizes a need for in-storage data transfer among any number of virtual machines. The next section considers the purposes served by such communication.

Generalized Management Information System

Many intercommunications problems were addressed in the course of a joint study conducted by the IBM Cambridge Scientific Center, the MIT Sloan School of Management (Center for Information Systems Research), the MIT Energy Laboratory, and the New England Regional Commission, a Federal-New England States co-partnership. The system that grew out of that study, the *Generalized Management Information System (GMIS)*,[31-34] is mentioned here because of its use of a separate virtual machines architecture for communication within the same real system. It exemplifies the application of many trends discussed in this section. The virtual machines include:

- Interactive CMS virtual machines which use FORTRAN, PL/I and APL interpreters.
- End-user virtual machines with application-oriented software such as econometric, time series analysis, and modeling packages.
- Experimental query machines which run SEQUEL.[35]
- A data-base-manager virtual machine which provides access to an experimental relational data base system. It activates itself and interfaces to other machines when they are needed.
- Interface virtual machines which accept requests from interactive virtual machines (running APL, for instance) and link them to facilities such as a relational-data-base manager. The user is unaware of this link. The linking and communication process is hidden from the application programmer and terminal user, who asks for and receives data and services without involvement in the intercommunication processes.

When the joint study began in 1975, the use of virtual punches and readers for intermachine communication was unsatisfactory because of the associated I/O and system overhead. Then CMS minidisks were used to exchange data among virtual machines. But the greatest improvement came with the experimental SPY interface discussed above. SPY was used both for virtual machine management and for the transfer of data. Finally, in the concluding months of the study, VMCF was used for transferring data between certain virtual machines.

The primary objective of GMIS was to interconnect a wide variety of language processors, application programs, analysis tools, and data base structures. Interconnection is made as the terminal user logs on the interactive virtual machine and decides what tools and resources are needed. This computational environment has been called an "ad hoc" decision support system (DSS).[36] The intercommunication capabilities and interface virtual machines were vital to the utilization of existing application programming and language processors without substantial modification. GMIS demonstrates the traditional virtual machine ability to accept diverse software environments without changing the software domains, even while providing sophisticated intercommunication and access to programming and data structures unknown to the application-oriented end user.

Virtual Control Storage

Research on virtual machine intercommunication continues. C. R. Attanasio of the IBM Thomas J. Watson Research Center has developed an experimental extension to VM/370 called *Virtual Control Storage* (VCS), a protected, fast-access execution and data domain for virtual machines.[37]

VMCF and VCS may appear to overlap when viewed strictly in terms of data transfer. Their architecture differs considerably,

Table 5 Performance comparisons of several virtual machine communications methods (times in seconds)[1]

Method	*No. of records*	*Send/ receive*	*Average time to send*	*Average virtual CPU time*	*Average total CPU time*	*Average no. of SIO's*
Spool	100	send	2.0	0.030	0.156	13
(virtual		recv	—	0.094	0.350	20
punch to	1000	send	11.2	0.252	1.010	104
reader)		recv	—	0.752	2.414	111
	10 000	send	114.2	2.494	9.448	1006
		recv	—	7.492	23.547	1013
DASD	100	send	3.2	0.060	0.210	47
(shared CMS		recv	—	0.050	0.220	38
minidisks)	1000	send	14.4	0.248	0.796	229
		recv	—	0.238	0.752	219
	10 000	send	102.8	2.170	6.338	2033
		recv	—	2.086	6.008	2023
VMCF[2]	100	send	0.8	0.004	0.030	1
		recv	—	0.006	0.056	1
	1000	send	1.4	0.024	0.140	11
		recv	—	0.032	0.190	10
	10 000	send	13.0	0.256	1.230	111
		recv	—	0.318	1.412	106

[1]These benchmarks were conducted by Clifford H. Avey at the IBM Cambridge Scientific Center.
[2]External interruptions for VMCF are counted as SIO's.

however, in that VMCF provides an asynchronous transfer mechanism between distinct virtual machines, whereas VCS provides synchronous data transmission between separate domains in the *same* virtual machine. Also, VMCF employs hardware storage protection and storage-to-storage transfer, whereas VCS relies on restricted addressability and segment sharing. VCS does not use an asynchronous communication-like protocol, but rather a synchronous instruction-like protocol. And the VCS program is able to modify areas of virtual storage and also general-purpose registers and the program status word. VCS applies, then, to far more than data transfer.

To place some of the communications techniques and developments in perspective, Table 5 cites several benchmarks that compare virtual punch-to-reader transfer, shared minidisk DASD, and VMCF, which uses storage-to-storage transfer. The benchmarks were conducted at the Cambridge Scientific Center using 80-byte records and sending and receiving virtual machines.

VM/370 networking

This discussion of inter-virtual-machine communication concludes with real-machine networking for VM/370, including transparent communication between virtual machines on physically separate real machines. Using RSCS as a base, peer CPU-to-CPU networking has been provided between systems connected on dial-up and leased lines or real channel-to-channel adapters. This capability is provided in the Network Job Entry/Network Job In-

terface (NJE/NJI) programming packages.[38] The VM/370 component is called VNET.[27] It provides peer connection to other CPU's (rather than the master-slave relationship in remote job entry). When those CPU's also run VNET, there is inter-virtual-machine communication between multiple real machines.

This approach, used extensively within IBM,[39] literally broadens the horizons for potential usage of virtual machines. In an experimental application at the Cambridge Scientific Center, for example, System/370 Models 158 and 115 were connected by a channel-to-channel adapter. Both ran VM/370, but the Model 115 executed a substantially smaller nucleus (approximately 100K bytes).[40] The experiment was meant to evaluate VM/370 in such an environment, and also to use the Model 115 to simulate a front-end processor by transmitting data to and from the Model 158. Both CPU's executed VNET and used it as their communications vehicle.

It is interesting to note that VNET's architecture necessitated no changes to CP and that it accommodated itself to a very small real machine environment without change. VNET resided in each of these two real machines. Because VNET supports CTCA and communications links for virtual-machine-to-virtual-machine protocols, the Model 115 just as easily could have been remote from the Model 158, to establish a concentrator or distributed processing application environment.

VI. Further use of operating system assist

The hardware assist philosophy, as discussed above for VM/370, is now provided in some other operating systems. Brief discussion is included here to balance the prior discussion and enable the reader to see that assist implementation is hardly limited to VM/370. Specifically, OS/VS1 benefits from a hardware assist on some models of System/370,[41] as does the MVS System Extensions Program Product for processors that support the System/370 Extended Facility.[42]

The hardware assist for OS/VS1 is called *Extended Control Program Support:VS1 (ECPS:VS1)*.[24,43] ECPS:VS1 and ECPS:VM can simultaneously reside in the writable control storage of a real system and provide a hardware assist to their respective control programs. For example, VS1 can be executing as a virtual machine under VM/370 and benefit from ECPS:VS1. CP and other virtual machines can benefit from ECPS:VM when they have control. ECPS:VM and ECPS:VS1 are not necessarily cumulative in their total effect on system performance, however. For instance, there is some overlap in ECPS:VS1 function and VS1 handshaking.[44]

The MVS System Extensions Program Product supports the System/370 Extended Facility. It can enhance system performance through new privileged instructions, path length reduction, and a variety of internal implementation changes to the system. Of the 14 new instructions defined by the System/370 Extended Facility, 12 are provided solely to assist MVS.[45]

While the assist hardware now has relevance for MVS on a real machine, there are also implications for MVS execution in a virtual machine. VM/370 SEPP accommodates and improves MVS performance in a virtual machine in part because the System/370 Extended Facility provides for execution of the new MVS privileged instructions by the assist hardware when MVS SEPP is running in a virtual machine. This capability is called the virtual machine extended facility and is specified through control register 6.[46] In addition, changes have been made to CP which improve MVS performance. This development might be called "partial handshaking" since changes are one-sided—only CP has been changed, in that it provides new MVS console SET commands: SET STM and SET STBY.

Together, these new commands allow CP to share shadow page tables[7,12,18] among the multiple virtual address spaces of a single MVS virtual machine when, in fact, the address space they point to is common. This will be the case in an MVS virtual machine for the nucleus and common area at the top of the virtual address space managed by MVS. The commands bypass shadow tables when MVS is running in a virtual-equals-real mode in the virtual machine.

Finally, users of APL interpreters on some models of System/370 experience faster execution because of the APL microcode assist. APL emulation involves direct execution of APL statements by microcode that works with a new System/370 instruction, APL EMULATION CALL (APLEC). Further discussion here is unnecessary, other than to note that the APL assist yields the greatest performance improvement of all the assists discussed in this paper. Given its early availability, the APL assist served to demonstrate what an effective microcode assist can achieve, and it stimulated subsequent assist implementations.[47]

VII. Summary

Since 1972, the implementation of System/370 virtual machines has changed considerably, specifically in CP and in the virtual machine interface. Changes within the virtual machine environment range from the APL assist microcode to virtual machine handshaking with CP, to accommodations for MVS virtual machines.

All of these changes had the dual objectives of improving performance and improving the functions of individual virtual machines, CP, and the entire system. Operational improvements provided with handshaking address certain needs associated with the virtual machine environment.

Virtual machine isolation and integrity have been preserved, along with optional extended facilities and interfaces that foster intermachine communication when desired. DIAGNOSE and VMCF in VM/370 make such interfaces generally available to the application developer and systems architect. To an extent, *inter-virtual-machine communication has always existed—it is its growth and potential that are most relevant.*

Several explicit examples, such as RSCS, VNET, and GMIS, illustrate new approaches to providing function, facility, and communication. They hold significant promise for innovative future use of the virtual machine concept and increased sharing of data and programming.

VIII. Conclusions

One can envision future uses of virtual machines that will exploit a proliferation of multiple, interconnected real and virtual machines. While the interconnections may be local or communications-based, the logical view can remain the same. VM/370 attached processor support[48] can be viewed as a first step—for large systems and in a tightly-coupled context. Multiple logical and real systems (large or small) are appropriate and fertile fields for future experimentation and investigation.

Possible implementations

data base machine

A virtual machine controlled by CP could assume responsibility for data management and encryption while connected to peer processing machines. It is important to note that these "data" and "processing" machines can be in the same real machine complex or in separate real machines. Logically, the processing machines might transmit requests in blocks to the "back-end" machine, which would asynchronously manage the data and transmit blocks of data responses (in clear or encrypted form). This technique of transmitting blocks of data already has been employed in the IBM 3705 communications controller when it runs the network control program/virtual storage (NCP/VS) for communication between hosts and the 3705 with Systems Network Architecture (SNA).

The GMIS and VCS experimental systems show that function can be divided among separate virtual machines or address spaces. For virtual machines to become separate real machines seems a logical next step. The possibility of improved "processing-ma-

chine'' performance, coupled with enhanced abilities to interconnect and share data in a secure fashion, should easily motivate future experimentation.

distributed processing

The virtual machine philosophy and system intercommunication capabilities (local and remote, virtual and real) hold promise for solving problems of distributed processing. The reliability and relative security of virtual machine systems enhance their attractiveness as vehicles for such experiments. The need to remotely assess and manage performance, malfunction, and operational problems is critical to such systems. Management of such problems is facilitated by the hypervision of virtual machine control in VM/370, and it is practical, considering the peer network connections that are available today. The ''user-friendly'' and ease-of-use characteristics of VM/370[3] also hold potential for such distribution, without increased complexity for the distributed system user.

programmed operator

The concept of a disconnected virtual machine as a programmed operator has been implemented at many VM/370 installations.[49] At Cambridge, for example, this concept has proved useful for directory and password management, mail and message handling, and secure volume mounting. Special virtual machines can be activated automatically by the programmed operator at a specified time to accomplish performance monitoring, for example. Such programmed operators in separate real machines may well prove beneficial for small systems that cannot support the programmed operations aspects of ASP or JES3 multisystems. When combined with network linking or as part of distributed processing systems, the programmed operator addresses unattended operation and human factors problems of computer operation in a non-data-processing environment (such as an office).

personal computing

CMS has long demonstrated the productivity that can be achieved with a single virtual machine. And while CMS is implemented to run on System/370, this type of virtual machine may well prove effective when running on a different hardware base—whether in a small computer like the IBM 5110 or in an intelligent terminal.

ACKNOWLEDGMENTS

I wish to express my appreciation to D. N. Saul, R. P. Parmelee, R. Reynolds, L. H. Holley, G. C. McQuilken, and L. H. Seawright of the IBM Cambridge Scientific Center for their scrutiny of the manuscript and attention to technical and performance details that eluded me; to W. J. Doherty, C. R. Attanasio, and A. N. Chandra of IBM Research, who enlightened me on SPY, VCS, and virtual machine subsystems; to E. C. Hendricks of IBM Research, who first drew my attention to virtual machine architecture, its value for subsystems, and how it is used by RSCS and VNET; to A. G. Olbert of IBM's System Products Division, who

was a codesigner of ECPS:VM and who provided a valuable critique of this paper and enhanced my understanding of ECPS:VM; to Barbara McCullough of IBM's Data Processing Division, who wrote the *Virtual Machine Facility/370 Features Supplement* and has helped many, including myself, with her masterful description of VM/370; finally, to the students at MIT's Sloan School who raised the essential questions that prompted me to write this paper.

Appendix: CP functions

The following CP functions are assisted by the CPA instructions:

- Obtain free space from free storage area
- Return space to free storage
- Page lock
- Page unlock
- Decode the first channel command word (CCW) in a list; also decode subsequent CCW's
- Free CCW storage
- Dispatch a control block or virtual machine
- Locate virtual I/O control blocks
- Locate real I/O control blocks
- Translate virtual address and (if possible) test for shared page
- Translate virtual address and (if possible) lock the page
- Invalidate segment table; invalidate page table
- Entry into virtual machine dispatch
- Common CCW processing
- Untranslate the channel status word (CSW)
- Dispatch control block or virtual machine
- LINK (initiated by CP's SVC 8)
- RETURN (initiated by CP's SVC 12)
- Change shared page scan

CITED REFERENCES AND NOTES

1. R. A. Meyer and L. H. Seawright, "A virtual machine time-sharing system," *IBM Systems Journal* **9**, No. 3, 199–218 (1970).
2. R. P. Parmelee, T. I. Peterson, C. C. Tillman, and D. J. Hatfield, "Virtual storage and virtual machine concepts," *IBM Systems Journal* **11**, No. 2, 99–130 (1972).
3. L. H. Seawright and R. A. MacKinnon, "VM/370—A study of multiplicity and usefulness," *IBM Systems Journal* **18**, No. 1, 4–17 (1979, this issue).
4. VM/370 is able to run on System/370 Models 135, 138, 145, 148, 155II, 165II, 158, 168, 158AP, 168AP, and 158MP and 168MP (configured with asymmetric I/O), and on processors 3031, 3031AP, 3032, 3033, and 3033MP (configured with asymmetric I/O).
5. *IBM Virtual Machine Facility/370 Introduction*, IBM Systems Library, order number GC20-1800, IBM Corporation, Department D58, P.O. Box 390, Poughkeepsie, New York 12602.
6. *IBM Virtual Machine Facility/370: Operating Systems in a Virtual Machine*, IBM Systems Library, order number GC20-1821, IBM Corporation, Department D58, P.O. Box 390, Poughkeepsie, New York 12602.
7. *Virtual Machine Facility/370 Features Supplement*, IBM Systems Library, order number GC20-1757, IBM Corporation, Department 824, 1133 Westchester Avenue, White Plains, New York 10604.

8. The term *handshaking* normally has been used only with OS/VS1, since DOS/VS uses the DOS/VS-VM/370 Linkage Facility in the Advanced Function-DOS/VS Program Product.
9. *OS/VS2 MVS Overview*, IBM Systems Library, order number GC28-0984, IBM Corporation, Department D58, P.O. Box 390, Poughkeepsie, New York 12602.
10. *OS/VS2 MVS/System Extensions General Information Manual*, IBM Systems Library, order number GC28-0872, IBM Corporation, Department D58, P.O. Box 390, Poughkeepsie, New York 12602.
11. The packaging and availability of VM assist (apart from any ECPS considerations) is different for different CPU's. It is standard on System/370 Models 135-3, 138, 145-3, and 148, and on the 3031 processor. It is a no-charge optional feature on System/370 Models 135-0, 145-0, and 158, and it is an RPQ on Model 168 and on the 3032 and 3033 processors.
12. P. H. Tallman, R. A. Denson, T. A. Gilbert, J. M. Nichols, and D. E. Stucki, *Virtual Machine Assist Architecture Description*, Technical Report TR 00.2506, IBM Corporation, Poughkeepsie, New York (January 1974).
13. F. R. Horton, D. W. Wagler, and P. H. Tallman, *Virtual Machine Assist: Performance and Architecture*, Technical Report TR 75.0006, IBM Corporation, Poughkeepsie, New York (April 1974).
14. Control register 6 is used by the hardware assists and by CP as an interface for their various activities.
15. For a discussion of the specific instructions, see *IBM System/370 Principles of Operation*, IBM Systems Library, order number GA22-7000, IBM Corporation, Department D58, P.O. Box 390, Poughkeepsie, New York 12602.
16. For a summary of the architecture and a list of System/370 instructions, see R. P. Case and A. Padegs, "Architecture of the IBM System/370," *Communications of the ACM* **21**, No. 1, 73–96 (January 1978).
17. To consolidate the error recording file (LOGREC) wherever possible, for the entire VM system (since CP has overall responsibility for control of the real hardware), VM assist presents SVC 76 to CP in the real machine, instead of the virtual machine's operating system. Most operating systems use SVC 76 to request the recording of certain error conditions (CPU and I/O) on their LOGREC files. When an interruption occurs, CP analysis determines whether CP can perform error recording and, if so, CP translates virtual device addresses into real ones and writes to LOGREC. When CP cannot handle SVC 76 (perhaps because of insufficient information from the virtual machine), it is reflected back to the virtual machine, whose operating system records the error information on its separate LOGREC file.
18. S. E. Madnick and J. J. Donovan, *Operating Systems*, McGraw-Hill Book Company, New York (1974). Chapter 9-5 offers a discussion of shadow tables for the general reader.
19. The virtual pointer list contains these relevant fields: addresses of the real segment table, virtual control registers, virtual program status word, and a virtual interruption-pending indication, as well as the assist control field (MICEVMA).
20. Relative batch throughput is the ratio of throughput on a native, real machine controlled by the specific operating system to throughput of the same job stream in a virtual machine controlled by VM/370.
21. A. G. Olbert, *Functional Description of Extended Control Program Support: VM/370*, Technical Report TR 01.2146, IBM Corporation, Endicott, New York (May 1978).
22. *IBM Virtual Machine Facility/370 System Extensions General Information Manual*, IBM Systems Library, order number GC20-1827, and *IBM Virtual Machine Facility/370 Basic System Extensions General Information Manual*, IBM Systems Library, order number GC20-1828, IBM Corporation, Department D58, P.O. Box 390, Poughkeepsie, New York 12602.
23. The instructions are LPSW, PTLB, SCKC, SPT, SIO, STNSM, STOSM, STPT, TCH, and DIAGNOSE.

24. *OS/Virtual Storage 1 Features Supplement*, IBM Systems Library, order number GC20-1752, IBM Corporation, Department 824, 1133 Westchester Avenue, White Plains, New York 10604.
25. *Advanced Functions—DOS/VS General Information*, IBM Systems Library, order number GC33-6040, IBM Laboratory, Publications Department, Schoenaicher Strasse 220, D7030 Boeblingen, Germany.
26. *Advanced Functions—DOS/VS Design Objectives*, IBM Systems Library, order number GC33-6039, IBM Laboratory, Publications Department, Schoenaicher Strasse 220, D7030 Boeblingen, Germany.
27. E. C. Hendricks and T. C. Hartmann, "Evolution of a virtual machine subsystem," *IBM Systems Journal* **18,** No. 1, 111–142 (1979, this issue).
28. A. N. Chandra and A. M. Katcher, *VM/370 Intermemory Communications—Comparative Measurements*, Research Report RC 5820, IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598 (January 1976).
29. *IBM Virtual Machine Facility/370: System Programmer's Guide*, IBM Systems Library, order number GC20-1807, IBM Corporation, Department D58, P.O. Box 390, Poughkeepsie, New York 12602.
30. R. M. Jensen, "A formal approach for communication between logically isolated virtual machines," *IBM Systems Journal* **18,** No. 1, 71–92 (1979, this issue).
31. J. J. Donovan and H. D. Jacoby, "Virtual machine communication for the implementation of decision support systems," *IEEE Transactions on Software Engineering* **SE-3,** No. 5, 333–342 (September 1977).
32. J. J. Donovan, "A note on performance of VM/370 in the integration of models and data bases," *The Computer Journal* **21,** No. 1, 20–24 (February 1978).
33. J. J. Donovan, R. Fessel, S. G. Greenberg, and L. M. Gutentag, *An Experimental VM/370 Based Information System*, Technical Report G320-2107, IBM Cambridge Scientific Center, 545 Technology Square, Cambridge, Massachusetts 02139 (July 1975).
34. C. S. Chandersekaran, and K. S. Shankar, "On virtual machine integrity," and J. J. Donovan and S. E. Madnick, "Virtual machine advantages in security, integrity and decision support systems," *IBM Systems Journal* **15,** No. 3, 264–278 (1976). This exchange of letters deals with the problem of virtual machine integrity and the use of GMIS.
35. D. D. Chamberlin and R. F. Boyce, "SEQUEL: a structured English query language," *ACM SIGMOD Workshop on Data Description, Access and Control* (Ann Arbor), 249–264 (May 1974).
36. J. J. Donovan and S. E. Madnick, "Institutional and ad hoc DSS and their effective use," *Data Base* **8,** No. 3, 79–88 (Winter 1977).
37. C. R. Attanasio, "Virtual Control Storage—security measures in VM/370," *IBM Systems Journal* **18,** No. 1, 93–110 (1979, this issue).
38. VM/370 Networking Program (No. 5799-ATA, PRPQ No. PO9007). This program was written by T. C. Hartmann and E. C. Hendricks, who extended their earlier work on RSCS.
39. The NJE/NJI networking packages have been used extensively in IBM's development and scientific laboratories, both during development of the programs and after their release. As of this writing, approximately 240 CPU's can communicate over leased lines or channel-to-channel adapters from Germany to California. For more information see R. P. Crabtree, "Job networking," and R. O. Simpson and G. H. Phillips, "Network job entry facility for JES2." *IBM Systems Journal* **17**, No. 3, 206–220 and 221–240 (1978).
40. The nucleus reduction work was done at the IBM Cambridge Scientific Center by L. Wheeler and J. Ravin. Dr. N. Sorensen conducted the VNET experiment.
41. Assist hardware for OS/VS1 is available on System/370 Models 135-3, 138, 145-3, 148, and 158, and on the 3031 processor.
42. *IBM System/370 Extended Facility*, IBM Systems Library, order number GA22-7072, IBM Corporation, Department D58, P.O. Box 390, Pough-

keepsie, New York 12602. This facility is available on System/370 Models 158-3 and 168-3 and on the 3031, 3032, and 3033 processors.

43. Specific functions performed by ECPS:VS1 are: page supervision (page enqueue and dequeue control blocks, short term fix/unfix, and invalidate page table entry); I/O supervision (CCW translation, local GET/FREE MAIN); storage management (GET/FREE MAIN of protected queue area or protected free queue element, SMF storage monitor maintenance); SVC handler (first-level handler code and, if needed, entry into SVC trace table); dispatch (the last portions of this routine); and trace recording (entries in trace table for SIO, I/O interrupt, and dispatch entry).
44. CCW translation and functions related to page table management performed by ECPS:VS1 microcode are eliminated when OS/VS1 handshaking is in effect. CP handles these functions and is helped by ECPS:VM.
45. The 12 instructions comprise four lock handling instructions, six tracing instructions, and page fix and SVC instructions.
46. See Figure 1 for the layout of control register 6. The virtual machine extended facility is supported only by the VM/370 System Extensions Program Product (Program No. 5748-XE1) and is controlled by bit 29 in control register 6. For its use to be valid, the MVS virtual machine must be in the virtual supervisor state and the real machine in the problem state. VM assist can be active at the same time. CP simulates the two new instructions—INVALIDATE PAGE TABLE ENTRY (IPTE) and TEST PROTECTION (TPROT)—which are part of the new architecture but not MVS-specific.
47. The APL assist concept is not unique with VM or CMS. APL assist features are available for the VSAPL interpreter supported by the VSPC Program Product (which runs under DOS/VS, OS/VS, and CMS), as well as for the APL/CMS interpreter which runs only on CMS systems. For additional information, the reader is urged to consult A. Hassitt and L. E. Lyon, "An APL emulator on System/370," *IBM Systems Journal* **15,** No. 4, 358-378 (1976).
48. VM/370 supports attached processors for System/370 Models 158AP and 168AP and for the 3031AP; and for Models 158MP and 168MP and the 3033MP when configured for asymmetric I/O. For more information on attached processor support, see L. H. Holley, R. P. Parmelee, C. A. Salisbury, and D. N. Saul, "VM/370 asymmetric multiprocessing," *IBM Systems Journal* **18,** No. 1, 47-70 (1979, this issue).
49. W. J. Doherty and R. P. Kelisky, "Managing VM/CMS systems for user effectiveness," *IBM Systems Journal* **18,** No. 1, 143-163 (1979, this issue).

Reprint Order No. G321-5085.