Development of a High Performance Virtual Machine System and Performance Measurements for it

HIDENORI UMENO*, KAZUHIKO OHMACHI*, AKIRA HINO*,
and JUNICHI IMURA**

Methods to enhance performance of more than one Virtual Machine in a Virtual Machine System (VMS) are presented. These include the implementation of V=Resident Virtual Machines, where memory is resident and real addresses are continuous, and of a fast process for Input/Output simulation (Fast I/O simulation).

A VMS's practicality depends on its performance, which is decreased by its CPU overhead. It is shown that a VMS's I/O simulation overhead is dominant within its total CPU overhead. Fast I/O simulation reduces the I/O simulation overhead.

Development and use of V=Resident Virtual Machines and Fast I/O simulation in a VMS has been undertaken on a HITAC M series computers, and performance measurements were obtained for this report.

Performance data which confirm the effectiveness of Fast I/O simulation are presented. Fast I/O simulation reduces a VMS's CPU overhead by about 40%~50% and significantly reduces elapsed time expansion for V=Resident Virtual Machines and for V=R Virtual Machine. Memory of V=R Virtual Machine is resident and its virtual addresses=real addresses except in the prefix area. As a result, performance improvement supports more than one High Performance Virtual Machine and makes it possible to apply a VMS to gradual system transition for high traffic on-line systems.

1. Introduction

A Virtual Machine (VM) is a functionally equivalent copy of a real host computer in which a statistically dominant subset of a virtual processor instructions execute directly on the real host processor [1]. A Virtual Machine System (VMS) consists of many VMs which are running concurrently. A Virtual Machine Control Program (VMCP) is a control program of a VMS.

Each user of a VMS can select a different Operating System (OS) because different OSs can run concurrently in different VMs. Virtual Machines are commonly used for system development and testing, and to obtain the services of more than one OS from a single real host machine.

As an example of the latter, VMs are used for gradual system transition. This means that by serving two different OSs (i.e. an old OS and a new OS) concurrently in a single real machine, users can gradually transfer from the system under the old OS to the system under the new OS. It costs users a vast sum of expense over a short term to convert all programs and files of the old system at once, when they could be used without conversion for some time. Such conversion also brings about reduced reliability. Users can avoid these problems by using VMs for gradual system transition. In on-line systems, such as for banking and information retrieval, gradual system transition is often required.

*Systems Development Laboratory, Hitachi Ltd.
**Hitachi Software Works.

The practicality of VMS is dependent on its CPU overhead and so, much effort have been made to reduce this overhead [3, 4, 6]. Representative examples of these are VM Assist (VMA) of IBM's VM/370 [3], IBM's VM/370 System Extensions (SE) [4], and VMS of Hitachi Central Research Laboratory (HCRL) [6]. The VMA of VM/370 is a firmware implementation designed to enhance the execution of privileged instructions and supervisor calls associated with VMs. In most cases effects of the VMA are remarkable, and it reduces VMCP's CPU overhead by 69% to 89% for some benchmark jobs [3].

In IBM's VM/370 SE and the VMS of HCRL, steps are taken in order to reduce the CPU overhead caused by switching virtual storage of an OS in a VM. Namely, these steps attempt to reduce the CPU overhead of a VM running a multiple virtual storage OS, and they do not address CPU overhead reduction for a VM running a single virtual storage OS which rarely switches its virtual storage. Other methods taken in HCRL's VMS have considerably reduced the CPU overhead but the performance can be enhanced for only one VM (i.e. V=R VM: explained later).

In spite of these efforts, it is difficult to use VMS for gradual system transition for high traffic on-line systems. The reasons are as follows.

(1) In many cases, an OS used for an on-line system is a single virtual storage OS or a real storage OS, to decrease the OS overhead. In such cases, methods that reduce the VMS's CPU overhead caused by switching multiple virtual storage of an OS in a VM, have a little effects on the on-line

system.

(2) On-line systems processes issue much more frequent file I/Os than those for batch systems, and the heavy VMS's I/O simulation overhead causes serious performance degradation.

(3) At least two high performance VMs are needed because at least two on-line systems may run concurrently in a single real machine. There exists only one high performance VM in a traditional VMS, however.

In order to use the VMS for gradual system transition for on-line systems, the VMS's CPU overhead has to be greatly reduced for more than one VM to guarantee proper response time for high traffic on-line systems. Therefore, several reduction methods are proposed here for an environment where necessary memory is available.

(a) In order to support more than one high performance VM, V = Resident VMs are introduced. The V = Resident VM has resident memory, real addresses which are continuous.

(b) Fast I/O simulation that greatly reduces I/O simulation overhead for normal I/O process is proposed and developed.

Performance data will be presented in Sec. 5 and it will be shown that VMS's performance is improved enough for the above-mentioned purpose.

2. Virtual Machine System (VMS)

Virtual Machine System gives mutiple users execution environments that have the same architecture as a real host computer. The architecture of a real computer, as discussed here, has the following characteristics.

(1) Central Processing Unit (CPU)

(a) Two processor states, called a supervisor state and a problem state.

(b) Privileged instructions, which can be executed only in the supervisor state.

(c) A means of addressing memory called segmentation and paging.

(d) A dynamic address translation feature, which translates virtual memory address into real memory address.

(e) A prefix area, which is in real memory and contains control information for various hardware interruptions.

(2) I/O channel units (simply called channels)

(a) I/O operations are started by CPU Start I/O instruction, and the Condition Code is set to indicate whether the I/O operations are started normally or not.

(b) Channels work concurrently with the CPU after normal start.

(c) Chennels request I/O interruptions to the CPU in order to signify normal or abnormal termination of the I/O operations.

(d) The CPU has I/O masks which indicate whether the CPU is enabled for the I/O interruption. If the CPU is enabled, an I/O interruption occurs, and control information of the I/O interruption is stored in the prefix area.

(e) Channels execute CCWs (Channel Command Words) which are commands given to the channels by the CPU. A CCW's execution sequence is called a channel program. A train of CCWs which are physically contiguous is called a channel program segment. Each segment is connected by branches in the channel program.

(f) A channel Status Word (CSW) is stored in the prefix area at the initiation of channel or at the I/O interruption. This CSW contains channel status information.

(g) Channels do not have a dynamic address translation feature. Namely, channels cannot directly execute CCWs on the virtual storage (virtual CCWs). Virtual CCWs have to be translated to functionally equivalent real CCWs by programs. This process is called CCW-translation.

(h) Channels have an indirect addressing feature. Channels can execute a CCW, data address of which is address of an area that contains data buffer addresses. This area is called Indirect Data Address Words (IDAWs).

VMs also have the two processor states described for CPUs above, complete instruction sets, storage, and prefix areas and so forth. The mechanisms used by VMCP to implement a VM fall into three categories: processor simulation, memory simulation, and I/O simulation [5].

Processor simulation is accomplished by running VM programs on a real processor. Nonprivileged state instructions are executed directly by the processor. VMCP traps privileged instructions and simulates them in software.

Memory simulation is performed by providing a virtual memory to each VM. This virtual memory appears to be a dedicated real memory complete with address translation hardware. An OS in a VM manages this memory as if it were real. Therefore, when an OS in a VM is a virtual storage OS (VOS), 3 level memory hierarchy is constructed, where:

level 1 memory:	real memory of a real processor;
level 2 memory:	memory of a VM, (An OS in a VM regards this as real memory);
level 3 memory:	virtual storage created by VOS in a VM.

Simulation of the dynamic address translation feature is performed by the VMCP using real hardware and special address translation tables (called shadow tables) that map level 3 memory addresses into level 1 memory addresses.

Input/Output simulation is performed in this way. The VMCP traps virtual Start I/O instructions, carrying out the CCW-translation and starting a real I/O operation on behalf of the user. After the real I/O interruption occurs, the VMCP translates real status information into virtual status information and reflects a virtual I/O interruption to the VM.

3. CPU Overhead in VMS

Major factors concerning CPU overhead in VMS and traditional reduction methods for the CPU overhead are described here. Norms for evaluating VMS's performance are introduced, and I/O simulation overhead dominancy is shown.

3.1 Major CPU Overhead Factors

(1) Overhead due to simulation: VMCP simulates privileged instructions issued by an OS in a VM, and also simulates the interruptions that belong to a VM.

(2) Overhead due to double management of system resources: An OS running in a VM manages its virtual resources and the VMCP also manages real system resources. In this environment, double management of system resources can occur. System resources include CPU, memory, I/O devices, etc. A typical example of this type is double paging. This means combination of paging for each level, that is, the VMCP performs paging between level 2 memory and level 1 memory while VOS in a VM performs paging between level 3 memory and level 2 memory.

(3) Overhead due to management of system resources: the VMCP manages its own memory and devices as an ordinary OS does.

3.2 Traditional Methods for Reducing VMS's CPU Overhead

Many means to reduce the CPU overhead have been undertaken before now. The traditional methods are listed below.

(1) Firmware assist: The assist emulates certain VMCP processes that simulate the instructions issued by VMs. A typical example is VM Assist of VM/370 [3].

(2) Handshaking with OS: An OS in a VM does not recognize that it is running in a VM environment. Handshaking changes this view for an OS, in that the OS is given the information that it is executing in a VM and can take certain actions in order to improve performance [3].

(3) Restrictions on usage of system resources: Real system resources can be dedicated to a VM to prevent the double management of system resources. As examples, devices may be dedicated to a VM and an area of real memory can be dedicated to a VM as in V=R VM. In the V=R VM, the level 2 memory is thoroughly resident in the level 1 memory and the level 2 memory addresses equal the level 1 memory addresses, except in the virtual prefix area.

Performance of the V=R VM can be enhanced, since the double paging and CCW-translation in the I/O simulation can be eliminated. However, only one V=R VM can exist in the single real host computer, because of its memory address characteristics V=R. A VM that is not the V=R VM is called a V=V VM.

This paper takes for granted that the above-mentioned traditional reduction methods have already been taken. In addition, it supposes that the following two methods have also been taken.

(1) VMA: VMA here contains functions equivalent to those of the VMA of VM/370 and additional 4 privileged instructions which are frequently used in on-line systems.

(2) Dedicated devices: All disc devices, magnetic tape devices for user files, and communication lines for user terminals are supposed to be dedicated.

An OS in a VM is supposed to be a single virtual storage OS, or a real storage OS, for the reasons described in Sec. 1. Therefore, CPU overhead due to switching OS's virtual storage in a VM can be neglected. However, the reduction methods described in Sec. 4 are also effective for multiple virtual storage OS in a VM.

Paging in VMS will increase CPU overhead to the extent where it can not be practically applied to on-line systems. Thus, an execution environment where real memory size is sufficiently large and the paging of VMCP almost does not occur, is considered as a premise in this paper.

Traditional reduction methods have considerably reduced the VMS's CPU overhead. The extent of reduction is not yet sufficient, however, and it is still difficult to apply VMS to gradual system transition for high traffic on-line systems, as described in Sec. 1.

3.3 Norms for Evaluating VMS's Performance

Performance of VMS is measured by an elapsed time expansion ratio (Er) and CPU overhead of VMS (Cr).

$$Er=\frac{dT_e^v}{T_e^b}\times 100 \quad (\%)$$

$$Cr=\frac{dT_c^v}{T_c^b}\times 100 \quad (\%)$$

$$dT_e^v=T_e^v-T_e^b$$

$$dT_c^v=T_c^v-T_c^b$$

T_e^v = elapsed time in a VM.

T_e^b = elapsed time in a real (i.e. bare) machine.

T_c^v = CPU service time in a VM.

T_c^b = CPU service time in a real (i.e. bare) machine.

The quantity dT_c^v consists of processing time for VMCP, and firmware assist such as VMA and expressed as follows.

$$dT_c^v=\sum_i\sum_j dT_c^{sv}(i,j)\cdot N_{ij}$$

$$dT_c^{sv}(i,j)=T_c^{sv}(i,j)-T_c^{sb}(i,j)$$

$T_c^{sv}(i,j)$ = simulation time of event i, under condition j in VMS.

Table 1 Simulation Overhead of each event (non-Fast I/O).

#	i	j	Events	Conditions	Simulation Overhead T_c^{sv}/T_c^{sb}
0	0	0	Start IO	V=R VM, normal case, no CCW-Translation	20~ 50
1		1		V=V VM, normal case	50~110*1
2	1	0	I/O Interruption	V=R VM, normal termination, no CCW-Translation	$\frac{T_c^{sv} \text{ of } \#2}{T_c^{sv} \text{ of } \#0}=0.8\sim1.0^{*2}$
3		1		V=V VM, normal termination	$\frac{T_c^{sv} \text{ of } \#3}{T_c^{sv} \text{ of } \#1}=0.6\sim0.7^{*1,*2}$
4	2	0	LPSW: Load Program Status Word	VMA is successful	2.91
5		1		Load Wait PSW	130~220
6	3	0	STOSM: Store then OR System Mask	VMA is successful	2.16
7		1	STNSM: Store then AND System Mask	I/O interruption pending	150~170
8	4	0	LRA: Load Real Address	VMA is successful	9.00
9	5	0	RRB: Reset Reference Bit	VMA is successful	5.13
10	6	0	SCKC: Set Clook Comparator	VMA is successful	7.75
11		1		External Interruption Pending	140~160

*1 Total Number of CCW$_s$=10, Number of CCW$_s$ of last channel prog. seg=5
*2 $T_c^{sb}=0$

$T_c^{sb}(i,j)$=execution time of event i, under condition j in a real machine.
N_{ij}=the number of occurrences of event i, under condition j.

Table 1 shows the simulation overhead of each event i, under condition j, and reveals that:

(1) If VMA fails or does not support event (i,j), then, simulation overhead T_c^{sv}/T_c^{sb} becomes 20~220 for it;
(2) If VMA succeeds for event (i,j), then simulation overhead T_c^{sv}/T_c^{sb} becomes 2~9 for it.

This means that if the VMA's feature is not supplied, the simulation overhead of privileged instructions is the primary factor of the VMS's CPU overhead.

Because VMA significantly reduces the simulation overhead [3], simulation overhead except for I/O simulation (events $i=0$, 1 in Table 1) becomes the secondary factor in the total CPU overhead.

3.4 I/O Simulation Overhead

Input/Output simulation overhead is expressed in this way.

$$dT_c^{IO}=\sum_j dT_c^{sv}(0,j)\cdot N_{0j}+\sum_j dT_c^{sv}(1,j)\cdot N_{1j}.$$

Table 1 shows that this I/O simulation overhead will be the primary factor of the total VMS's CPU overhead if the VMA's feature is available.

Input/Output simulation overhead ratio over total VMS's CPU overhead is defined as follows.

$$T_{cr}^{IO}=\frac{dT_c^{IO}}{dT_c^v}\times 100 \quad (\%)$$

This I/O simulation overhead ratio T_{cr}^{IO} for several benchmark jobs in Table 2 is shown in Fig. 1. This figure reveals that I/O simulation overhead forms 60% to 85% of the total VMS's CPU overhead. Namely, I/O simulation is dominant in the VMS's CPU overhead. Therefore, in the case of applying the VMS to on-line systems that frequently access many files, VMS's performance will be seriously reduced.

The measurements in Fig. 1 were conducted under the following conditions.

(1) The V=R VM and V=Resident VM were used.

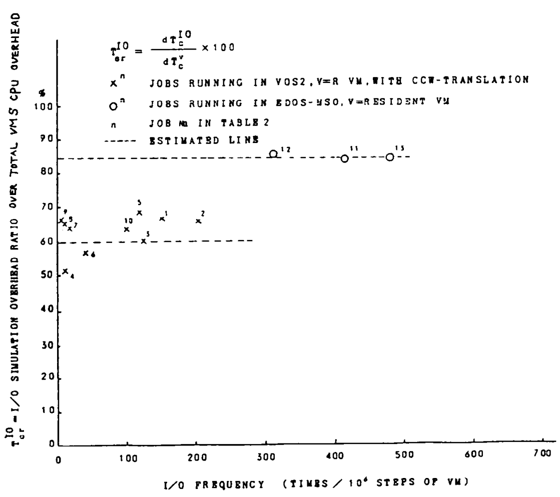

Fig. 1 Dominancy of I/O Simulation Overhead in total VMS's CPU overhead (non-Fast I/O).

The latter will be introduced in Sec. 4 as a high performance VM different from the V=R VM.

(2) In the V=R VM a single virtual storage OS (VOS2*[1]) was running and in the V=Resident VM a real storage OS (EDOS-MSO*[2]) was running.

(3) All the virtual channel programs were translated in the V=R VM and in the V=Resident VM.

(4) Fast I/O simulation which is a CPU overhead reduction method, and is described in Sec. 4, was not used.

4. Methods for Supporting More Than One High Performance VM

Two methods have been developed to support more than one high performance VM:

(1) V=Resident VMs;

(2) Fast I/O simulation.

These methods are described in the following sections.

4.1 V=Resident VMs

At least two high performance VMs are required for implementing gradual system transition. The V=R VM will be used for that purpose, because double paging and CCW-translation overhead can be eliminated in it. However, only one V=R VM can exist in the system, as is stated in Sec. 3.2.

As high performance VMs different from the V=R VM, V=Resident VMs have been introduced. The V=Resident VM has level 2 memory that is thoroughly resident in the level 1 memory and has the following address characteristics, as illustrated in Fig. 2:

level 1 memory address

$$=\text{level 2 memory address}+\alpha$$

The displacement $\alpha(\neq 0)$ is a fixed page number, and given to each V=Resident VM at VMS's system generation.

The V=Resident VMs require shadow tables as before, however, they can eliminate double paging, since their memory is resident in real memory. Many V=Resident VMs can be defined as long as the real memory size permits. Moreover, CCW-translation overhead for the V=Resident VM can be reduced through its memory address characteristics. The CCW-translation overhead is dominant in I/O simulation time, as illustrated in Fig. 3-(a). For this reason, the VMS's CPU overhead cannot be decreased enough, unless the CCW-translation overhead is reduced. Details concerning reduction of the CCW-translation overhead are described in Sec. 4.2.2.

*[1]Virtual Storage Operating System 2: OS on the HITAC M series computers.

*[2]Extended Disc Operating System with Multi-Stage Operations: same as above.

4.2 Fast I/O Simulation

4.2.1 General I/O Simulation and Fast I/O Simulation

The I/O simulation process of VMCP in general cases, is very complex, because it contains not only the process for normal cases but also for abnormal cases. The latter contains many complex error recovery processes for various I/O devices. However, among general cases, normal cases occur more frequently than abnormal cases. In normal cases, I/O operations are started normally, then, they terminate completely and I/O termination interruption occurs.

Letters (a), (c) in Fig. 3 show the general I/O simulation process for normal cases and processing time for each processing phase which is calculated by its execution steps. This figure reveals that:

(1) The CCW-translation is the primary overhead of Start I/O simulation (Fig. 3-(a)-(2));

(2) Reflecting the I/O interruption in the VM is the primary overhead of the I/O interruption simulation (Fig. 3-(c)-(2), (4)).

In Sec. 3.4 it is stated that the I/O simulation is dominant in the total CPU overhead of VMS. In order to reduce I/O simulation overhead, new short cut modules have been created for normal processing in I/O simulation. They are called Fast I/O simulator and Fast dispatcher, and are illustrated in Fig. 4. Normal I/O simulation cases are always handled by the fast logic of the I/O simulation (i.e. Fast I/O simulation). When exceptions (e.g. Start I/O Condition Code is not 0, or I/O errors, or I/O devices that Fast I/O does not process) are detected, control is transferred to the general logic and the remainder of the process is entrusted to it.

4.2.2 Reduction Methods in Fast I/O Simulation

4.2.2.1 The Fast I/O Simulation Logic

In Fast I/O simulation logic, except for CCW-transla-

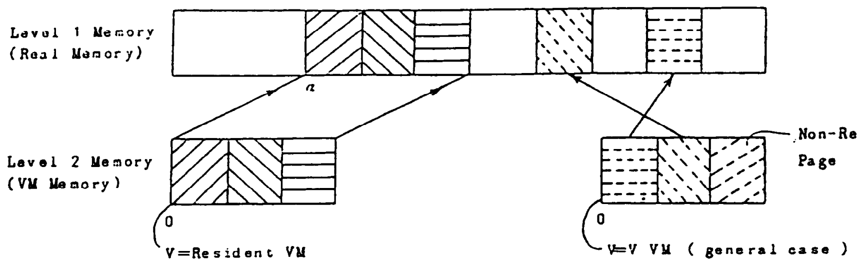

Fig. 2 Memory Mapping of V=Resident VM, and V=V VM.

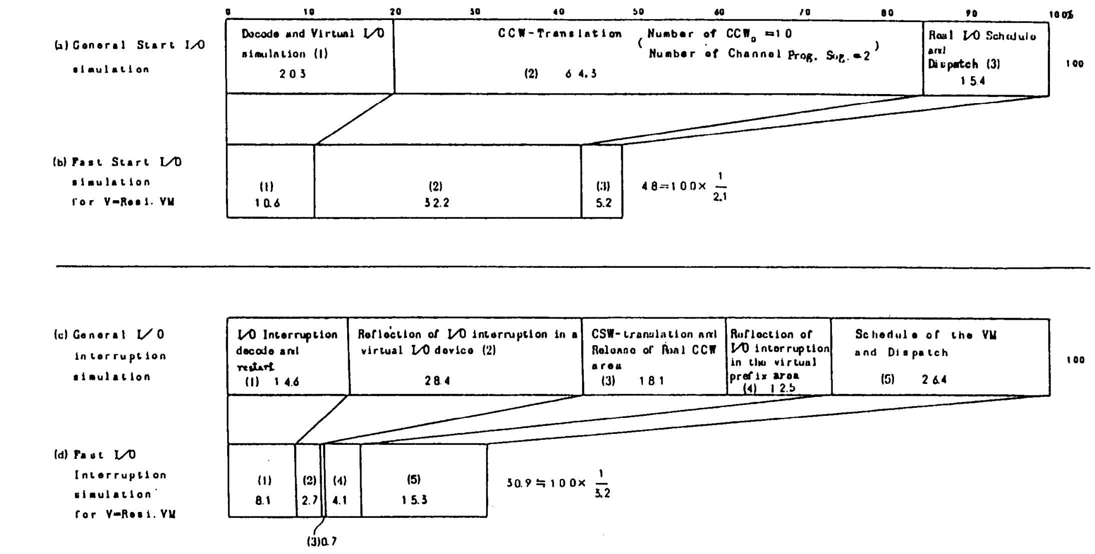

Fig. 3 I/O simulation time for General I/O vs. that for Fast I/O.

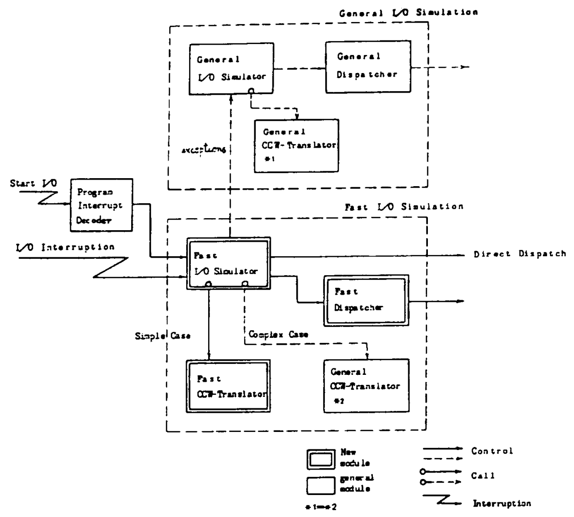

Fig. 4 Relation between General I/O Simulation and Fast I/O Simulation.

tion, the following I/O simulation overhead reduction methods are taken.

(1) Restrictions on the device usage: For the logical simplicity Fast I/O simulation is applicable only to dedicated devices (Magnetic discs, tapes, lines, etc.) which are frequently used in on-line systems.

(2) Static acquisition of various I/O control areas: In order to reduce CPU overhead, various I/O control areas are acquired at system initiation of VMCP or VMs. Their fields' value are also fixed at the initiation if possible. The reduction in Fig. 3-(b)-(1) is largely due to this. The general I/O logic usually acquires them on demand.

(3) Local Memory management: The Fast I/O logic manages its own I/O work areas for itself because of elimination of the global VMCP memory management overhead. The reduction in Fig. 3-(d)-(3) is largely due to this.

(4) Fast translation from real to virtual addresses: The real addresses of I/O control areas, the CCW address in the CSW, etc. have to be translated into the virtual addresses in a virtual I/O interruption simulation process. This translation is made faster with the I/O control table which contains the correspondence of the real addresses to virtual addresses. The reduction in Fig. 3-(d)-(2), (3) is largely due to this.

(5) Fast I/O interruption reflection: In the general logic, the general I/O simulation reflects the I/O interruption in a virtual device, including a virtual channel, and the general dispatcher reflects it in the virtual prefix area and clears the interruption information in the virtual device if the VM is enabled for it. The Fast I/O simulator contains the I/O simulation logic, and a simple dispatching logic (Fig. 4). Therefore, the I/O interruption can be directly reflected in the virtual prefix area without being reflected in a virtual device if the VM is enabled for it. The reduction in Fig. 3-(d)-(2), (4) is largely due to this.

Fast I/O simulation is applicable not only to the V=R or V=Resident VM but also to the general V=V VM. This Fast I/O simulation has been implemented and is estimated to reduce the I/O simulation time except CCW-translation time by 50% to 75% in the V=R or V=Resident VM, and by 37% to 44% in the general V=V VM.

4.2.2.2 General CCW-Translation Process

The CCW-translation is the primary overhead in the I/O simulation time as illustrated in Fig. 3-(a). When all the pages of a VM are resident in the level 1 memory, major factors of the CCW-translation overhead become as follows. Each time a virtual CCW is accessed,

(1) VMCP needs to look up the address translation table, since the next CCW in a channel program may cross the page boundary.

(2) VMCP needs to look up the address translation table when it translates a data address of a virtual CCW into a real address.

(3) VMCP has to check if the data buffer may cross the page boundaries and if so, it needs to create indirect data address words (IDAWs).

(4) The general CCW-translation handles general channel programs in which their branches may be very complex. For this reason, processing their branches in the general CCW-translation becomes complex.

4.2.2.3 Fast CCW-Translation for the V = Resident VM

The V = Resident VM has the resident level 2 memory and has the following feature.

level 1 memory address

$$= \text{level 2 memory address} + \alpha$$

By this feature the CCW-translation is significantly simplified and made fast as follows.

(1) As real memory address of the V = Resident VM is continuous, VMCP need not check page boundaries of a virtual channel program.

(2) Data address of a virtual CCW is translated into real address only by adding the displacement. Therefore, VMCP need not look up the address translation table.

(3) VMCP need not create IDAWs even if data buffer of a virtual CCW may cross page boundaries for the same reason as (1).

Moreover, the following reduction methods are taken.

(4) Only simple channel programs in which their branches are simple can be processed by Fast CCW-translation. For example, the branch to more than one channel program segment from a single channel program segment is not permitted. The translation of complex channel programs is entrusted to the general CCW-translation. By this restriction, the translation process for their branches is made very simple and fast.

(5) Real CCW areas are acquired as free-lists at the system initiation and are locally managed in the Fast I/O simulation. This reduces the global memory management overhead.

This Fast CCW-translation logic is only applicable to the V = Resident VMs or the V = R VM, and does not require OSs to be modified. The relation between Fast I/O simulator and Fast CCW-translator is shown in Fig. 4.

This Fast CCW-translator was implemented and is estimated to result in about 50% reduction in the CCW-translation overhead. This effect was confirmed by measurements in which Fast CCW-translation was temporarily invalidated.

4.2.2.4 Fast CCW-Translation by Handshaking with OSs

Start I/O instructions issued by an OS in a VM give VMCP nothing but the first CCW address and I/O address which VMCP uses to translate virtual CCWs. If an OS in a VM gives VMCP certain information on the CCW-translation at issuing Start I/O, the CCW-translation process will become simpler and faster in the V = Resident VM. This information contains the last CCW address, and the number of words in IDAWs, etc. It is estimated that this handshaking will cause approximately 67% reduction in the CCW-translation overhead in the V = Resident VM. However, because this method requires OSs to be modified, this is not implemented at present.

Letters (b), (d) in Fig. 3 show that the I/O simulation time is estimated to be decreased to $1/2.1 \sim 1/3.2$ for the V = Resident VM. Almost the same effects are estimated for the V = R VM.

5. Performance Improvement Measurement

The V = Resident VMs and the Fast I/O simulation have been developed in the VMS on the HITAC M series computers, and the performance has been measured on the HITAC M180.

5.1 Tools for Performance Evaluation

The following tools are used that are built in the VMS.

(1) Software monitor: This monitors the number of occurrences of events such as virtual privileged instructions, interruptions, and so forth.

(2) Hardware monitor: This monitors the CPU service time in the supervisor state or in the problem state, the instruction counters, not in Buffer counters, not in TLB (Translation Look Aside Buffer) counters, channel busy time, and so forth.

(3) Data analyzing program: this program sums up the data of monitors and prints out the results.

5.2 Hit Ratio Definition

In order to represent the applicability of the Fast I/O simulation hit ratios of it is defined as follows:

$$\text{Hit ratio of Fast Start I/O} = \frac{D}{C} \times 100 \quad (\%)$$

C = the total number of virtual Start I/O;

D = the number of virtual Start I/O which are completely handled by the Fast Start I/O simulation.

Hit ratio of Fast I/O interruption is defined in the same way. Average hit ratio of Fast I/O simulation is defined as a simple average of the above-stated two ratios.

Hit ratio of Fast CCW-translation is defined similarly:

Table 2 Benchmark Jobs.

Job No.	Measurement Conditions	Outlines	CPU Utilization (%)	I/O frequency times/10^6 steps of VM		Remarks
				non-Fast I/O	Fast I/O	
1	VOS2	Generation of 100,000 data	13.1	149	152	
2	V=R VM with	Sorting that data	13.1 [*3]	199	199	[*1] Magnetic Tape
3	CCW-translation Single VM	transfer of 61 MB[*2] programs from MT[*1] to DISC	15.7	122	123	[*2] Million Bytes
4	Single Job	Compiling 12 PL1 programs	62.8	10	10	[*3] Under Single VM
5		446[*5]	18.1	117	102	[*4] Under Real Machine
6		52	43.1	39	36	
7		Compile-Link-Go of Fortran programs 700	54.8 [*4]	15	14	[*5] the number of of source program coding steps
8		141	75.0	8	7	
9		219	81.4	5	4	
10		321	25.9	98	64	
11	EDOS-MSO	Same as Job No. 1	6.2	413	424	
12	V=Resident VM Single VM	Same as Job No. 2	5.3	311	312	
13	Single Job	Same as Job No. 3	4.9	477	476	

$$\text{Hit ratio of Fast CCW-translation} = \frac{F}{E} \times 100 \quad (\%)$$

E=the total number of the Fast CCW-translation requests;

F=the number of CCW-translation requests which are completely handled by the Fast CCW-translator.

These hit ratios are calculated from data of the software monitor.

5.3 Benchmark Jobs and Independent Programs

Table 2 shows benchmark jobs. CPU utilization and I/O frequency ranges from 4.9% to 75.0% and from 4 to 476 times/10^6 steps of VM respectively. In addition, independent programs were developed and used to measure the I/O simulation time. It contains only one privileged instruction: Start I/O. The measurements has confirmed the effect of the Fast I/O simulation illustrated in Fig. 3.

5.4 Performance Improvement

Table 3 summarizes the effects of Fast I/O simulation for the benchmark jobs in Table 2. Each item is discussed here as follows.

(1) Hit Ratio of Fast I/O simulation

Fast I/O simulation handles only the normal cases. Therefore, if abnormal cases occur frequently, the hit ratios will be decreased. At present, the occurrences of unit check (some hardware errors in real devices) are the main cause that reduces the hit ratios (Job No. 5 to No. 10 in Table 3). Unless the unit check occurs, the hit ratios are close to 100% (job No. 1 to No. 4, job No. 11 to No. 13 in Table 3).

(2) Reduction in supervisor state time

Figure 5 reveals that if the averrage hit ratio of Fast I/O simulation $\geqq 90\%$ then, supervisor state time is reduced by about 50%~60% and the CPU overhead of VMS is also reduced by about 40%~50% (Fast I/O vs. non-Fast I/O).

(3) Reduction in Elapsed time expansion

Table 3 shows that Fast I/O simulation significantly reduces the elapsed time expansion ratio.

(4) Mean Instruction Execution Time of VMCP

Table 3 shows that the Mean Instruction Execution Time (MIET) of VMCP is increased a little by Fast I/O simulation. This is because the number of short instructions (for example, those executed in one or two machine cycles) becomes smaller by the significant elimination of CPU steps of I/O simulation by Fast I/O logic.

On the other hand, the not in TLB ratio of VMCP is decreased a little by Fast I/O simulation. In addition, the not in Buffer ratio of VMCP is decreased also, when the Average Hit Ratio of Fast I/O simulation >99% as illustrated in Fig. 6. This means that the locality of program modules in Fast I/O simulation is higher than that in general I/O simulation.

(5) VMA's overhead

VMS's overhead is calculated by its simulation time for each privileged instruction. The time is calculated

Table 3 Effects of the Fast I/O simulation.

Job No.	*1Reduction in supervisor statertime	*2Reduction in VMS CPU Overhead	*3$\frac{\text{OVMA}}{T_c^b}\times 100$ (%)		$E_r=\frac{dT_e^v}{T_e^b}\times 100$ (%)		Hit Ratio of		
			A*4	B*5	A	B	Fast Start I/O	Fast I/O Interrupt	Fast CCW Translation
1	54.6%	40.2%	28.3	30.5	—	—	99.9% *9(99.9%)	99.8%	100%
2	55.1	39.4	34.8	37.5	—	—	98.8 (99.0)	99.1	100
3	60.0	37.2	52.2	55.6	—	—	100 (100.0)	99.9	100
4	47.7	33.1	3.8	3.8	—	—	99.0 (97.0)	95.0	100
5	32.9	28.5	16.9	15.8	23.5	13.7	77.4 (70.8)	64.2	99.7
6	41.2	34.1	11.2	9.6	22.5	10.0	88.2 (85.9)	83.5	99.5
7	32.2	25.3	3.6	3.6	11.8	4.4	83.1 (77.8)	72.5	99.5
8	37.5	30.6	1.5	1.5	7.4	2.1	84.5 (79.6)	74.7	99.4
9	37.8	28.3	0.9	1.0	5.2	1.7	85.5 (81.1)	76.6	99.3
10	49.6	45.8	16.4	11.7	20.7	6.9	85.4 (81.1)	76.7	99.4
11	46.0	45.8	1.6	1.7	11.0	0.4	99.9 (99.7)	99.5	100
12	47.8	47.0	1.7	1.7	3.8	0.3	99.4 (99.4)	99.4	100
13	47.2	46.9	2.8	3.0	8.0	1.7	100 (99.1)	98.1	100

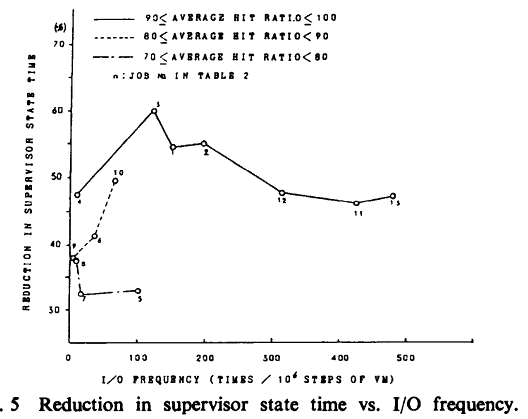

Fig. 5 Reduction in supervisor state time vs. I/O frequency. Several intervals of average hit ratio for Fast I/O shown.

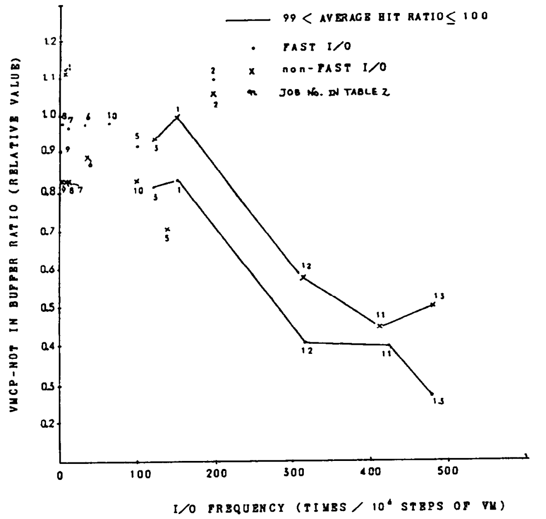

Fig. 6 VMCP-not in buffer ratio (NIBR) vs. I/O frequency. Effects of Fast I/O shown.

based on the trace of micro-program coding. This calculated time is used in Table 3, since it expresses the time of standard process, and measured time may express that of special process. Table 3 shows that the VMA's overhead (OVMA) over real CPU service time (T_c^b) will amount to 30%~55% in the case of I/O bounded jobs of VOS2 (Job No. 1~No. 3). This VMA's overhead changes depending on if Fast I/O simulation is used or not, since the behaviour of an OS in a VM also changes depending on that.

5.5 Memory Increase in Fast I/O Simulation

Real memory is required for level 2 memory of the V=R VM or V=Resident VMs. In addition, the following memory increase is required for various I/O control areas for Fast I/O simulation.

Figure 7 shows the memory increase in Fast I/O

VMCP-MIET*6 relative value		VMCP-NIBR*7 relative value		VMCP-NITLBR*8 relative value		Remarks
A	B	A	B	A	B	
1	1.187	1	0.834	1	0.798	*1 Fast I/O vs. non-Fast I/O
0.996	1.217	1.055	1.091	1.105	0.968	*2 Fast I/O vs. non-Fast I/O
0.957	1.135	0.942	0.816	0.871	0.468	*3 VMA overhead
1.024	1.202	1.116	1.123	0.976	0.742	*4 non-Fast I/O
1.045	1.094	0.711	0.926	1.006	0.855	*5 Fast I/O
1.083	1.143	0.899	0.982	1.036	0.897	*6 Mean Instruction Execution Time
1.070	1.120	0.837	0.974	1.017	0.861	*7 Not In Buffer Ratio
1.073	1.127	0.839	0.981	0.986	0.834	*8 Not In TLB (Translation Look aside Buffer) Ratio
1.072	1.117	0.832	0.912	0.959	0.788	*9 AVERAGE HIT RATIO
1.073	1.135	0.830	0.984	1.035	0.903	
1.030	1.112	0.448	0.295	0.823	0.500	
1.035	1.112	0.572	0.401	0.879	0.532	
1.030	1.101	0.499	0.267	0.839	0.669	

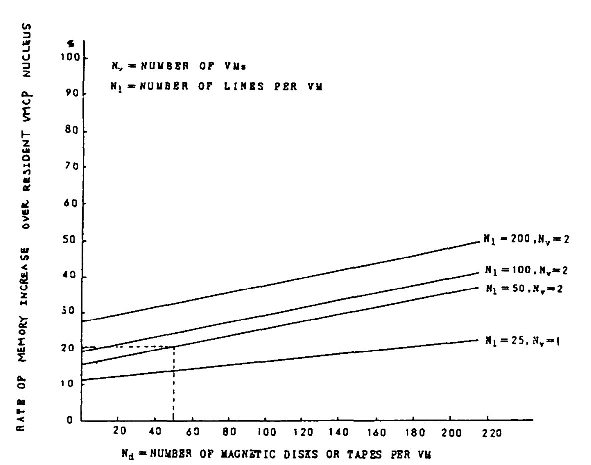

Fig. 7 MEMORY INCREASE in FAST I/O simulation.

simulation for the number of virtual devices of a VM. It shows that Fast I/O simulation needs more memory by about 20% than general I/O simulation logic, when $N_d = 50$, $N_l = 50$, $N_v = 2$. Here:

N_d = the number of discs or tapes per VM;

N_l = the number of communication lines per VM;

N_v = the number of VMs initiated.

This is because various I/O control areas are acquired at the system initiation in order to reduce CPU overhead in Fast I/O simulation.

6. Conclusions

The V = Resident VMs have been introduced and Fast I/O simulation method has been described. The effect of Fast I/O simulation has been confirmed by measurement. Fast I/O simulation causes about 50%~60% reduction in total supervisor state time of VMS and significantly reduces elapsed time expansion for the V = Resident VM and for the V = R VM.

Memory increment in Fast I/O simulation is about 20% for the 200 devices for users in the system.

This performance improvement supports more than one high performance VM (i.e. the V = R VM and the V = Resident VMs), and these VMs are now being used for gradual system transition for several on-line systems.

Acknowledgements

The authors express their sincere thanks to Mr. M. Kodaira and Mr. K. Yamamoto of Hitachi Software Works, and Mr. S. Takasaki of Systems Development Laboratory for their helpful comments and for their participation in technical discussions.

References

1. Goldberg, R. P. Architectural principles for virtual computer system, Ph. D. Thesis, Div. of engineering and Applied Physics, Harvard University, Cambrige, Ma, 1972.
2. Goldberg, R. P. Survey of virtual machine research, *Computer*

7, 6 (June 1974), 34–35.
3. MACKINNON, R. A. The changing virtual environment: Interfaces to real hardware, virtual hardware, and other virtual machines, *IBM SYST J.* **18**, 1 (1979).
4. IBM Virtual Machine Facility/370 System Extensions, *General Information Manual*, No. GC20–1827.
5. FRITZ, M. D. A Virtual Machine Emulator for Performance Evaluation, *C. ACM* **23**, 2 (Feb. 1980).
6. TAGUCHI, T., HORIKOSHI, H. and KURIHARA, J. Design and Experiments of a Virtual Machine System, *Information Processing Society of Japan*, **20**, 4 (July, 1979).

(Received August 4, 1980: revised March 9, 1981)