Regular Paper

Reusing TLB Entries for Virtual Machines in Processor Switching

Hidenori Umeno[†] and Hiroshi Ikegaya[††]

A method for reducing the not-in-TLB ratio (NITR) is presented in the environment of concurrently running multiple virtual machines (VMs), which are functional copies of real host computers. A VM contains at least one logical processor. The translation lookaside buffers (TLBs) contain pairs of virtual addresses and corresponding real addresses so that virtual storage addresses can be quickly translated into corresponding real storage addresses. The NITR is the ratio of the number of address translations outside the TLBs to the number of instructions executed. The VM CPU performance depends heavily on the NITR of the VMs. Conventionally, TLB entries for a logical processor are always purged when the running of that logical processor is switched to another real processor. This is because the logical processor may have issued non-signalling-purge-TLB-type instructions, which purge TLB entries only for that logical processor. The proposed method provides a real processor with a means for remembering that a logical processor of a VM has issued non-signalling-purge-TLB-type instructions, and for reusing its TLB entries for the logical processor even when the running of that logical processor is switched to another real processor, if the logical processor has not issued such instructions since the last time the TLB entries were purged. The proposed method purges TLB entries for a logical processor only when that logical processor has issued such instructions, and thus avoids excessive purging of the TLB entries for logical processors. It is effective for floating scheduling, in which any free real processors can run any ready logical processors, decreasing the NITR of VMs to 1/2–1/3 that of VMs in the conventional method. Consequently, it will reduce the VM CPU time by 9–12% of that in the conventional method.

1. Introduction

A virtual machine (VM) is a functional copy of a real host computer[1]. The real host computer directly executes most instructions of a VM. A virtual machine system (VMS) consists of multiple VMs, which can run concurrently. Different operating systems (OSs) can run in different VMs concurrently. A virtual machine monitor (VMM) is a control program of a VMS, allocating real processors to VMs and scheduling the VMs. A real processor can be active either in native mode or in VM mode. In native mode it runs a single OS, which manages all its resources. In VM mode it runs multiple OSs in VMs concurrently.

An OS provides user programs with virtual address spaces. The user programs run in their virtual address spaces. A real processor has a dynamic address translation feature (DAT)[11], which translates virtual addresses in the virtual address spaces into real addresses in real memory. The speed of the address translation greatly affects the speed of instruction execution, and thus the total system performance.

The DAT consists of address translation tables, which an OS creates and manages while running user programs, and translation lookaside buffers (TLBs), which are address translation hardware devices that contain combinations of virtual addresses and corresponding real addresses[11].

A real processor contains several TLBs; for example, it may have one for instruction addresses and another for operand addresses. It uses the TLBs to translate virtual addresses quickly into corresponding real addresses. This is called TLB translation. When it cannot find a current virtual address in its TLBs, a real processor looks in the address-translation tables, which are in main memory, for the virtual address. This is called table-look-in translation, and is tens of times slower than the TLB translation. The ratio of table-look-in translations to instructions executed, called the "not-in-TLB ratio" (NITR), greatly affects the speed of instruction execution.

A real multiprocessor system contains multiple real processors sharing main memory. Each processor has its own TLBs, which include entries for each processor. Methods for controlling TLBs in VM mode are somewhat different

† Software Development Department, General Purpose Computer Division, Hitachi, Ltd.
†† Development Department 1, General Purpose Computer Division, Hitachi, Ltd.

from those in native mode, because TLB entries contain VM identifiers or their equivalents[9],[10]. The TLBs may include entries for any VMs, because the VMs may run on any shared real processors. When it is in multiprocessor mode, a VM consists of multiple logical processors with a shared main memory area.

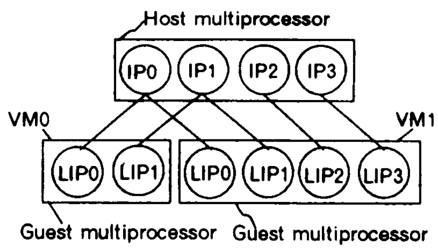
The NITR in VM mode tends to be higher than that in native mode. This decreases the VM CPU performance. The tendency is caused by excessive purging of TLB entries for logical processors. One typical and conventional case of excessive purging is as follows. OSs often use control instructions, which are called purge-TLB-type instructions, to directly purge TLB entries. Real processors tend to purge TLB entries excessively in order to simplify the simulation for those control instructions in VM mode. An example of such simplification is that real processors always purge TLB entries for logical processors when the VMM switches the dispatching of logical processors from one real processor to another real processor[9], whether or not the logical processors have ever issued the purge-TLB-type instructions.

To avoid increasing the NITR in VM mode, we present a new method for determining precisely whether or not TLB entries are valid: The method remembers that logical processors have issued purge-TLB-type instructions, and makes the hardware/micro-programs of real processors purge TLB entries for logical processors only if those logical processors have issued purge-TLB-type instructions. The method reduces the NITR in VM mode to 1/2–1/3 that in the conventional method, and will reduce the VM CPU time by 9–12% of that in the conventional method.

Section 2 explains conventional methods for scheduling real or logical processors, and Section 3 explains conventional methods for managing TLB entries and their problems. Section 4 proposes a new method for managing TLB entries for logical processors, and discusses performance improvement for the new method. Conclusion is presented in Section 5.

2. Conventional Methods for Scheduling Processors

A host multiprocessor system contains multiple real processors that share the system's main memory. Similarly, a guest multiprocessor system contains multiple logical processors that share the guest's (i.e., the VM's) main memory area. The VMM is a control program, allocating real processors to logical processors as follows.



IP: Physical (instruction) processor
LIP: Logical (instruction) processor

Fig. 1 Fixed scheduling.

2.1 Fixed Scheduling

This scheduling rigidly allocates real processors to logical processors. That is, the VMM makes the logical processors of a VM uniquely correspond to real processors. Those logical processors can be run only on the corresponding real processors. An example of this scheduling is shown in **Fig. 1**: The real host computer is a four-way multiprocessor, having IP0, IP1, IP2, and IP3, which are also called instruction processors. VM0 is a two-way multiprocessor, having LIP0, and LIP1, which are called logical processors. VM1 is a four-way multiprocessor, having LIP0, LIP1, LIP2, and LIP3. The VMM runs LIP0 and LIP1 of VM0 only on IP0 and IP1, respectively, and runs LIP0, LIP1, LIP2, and LIP3 of VM1 only on IP0, IP1, IP2, and IP3, respectively. Real processor IP0 is shared by logical processors LIP0 of VM0 and LIP0 of VM1. Real processor IP1 is shared by logical processors LIP1 of VM0 and LIP1 of VM1. Real processors IP2 and IP3 are dedicated to logical processors LIP2 and LIP3 of VM1, respectively.

2.2 Floating Scheduling

This scheduling enables any free real processors to run any ready logical processors. **Figure 2** shows an example, in which real host computer is the same as that in Fig. 1. VM_0, $VM_1, \cdots$, and VM_{N-1} are single-processor or multiprocessor VMs.

Real processors (IPs) are shared by logical processors (LIPs). The total number of LIPs of all the VMs is usually equal to or larger than the number of IPs of the real host computer. The VMM runs any LIPs on any free IPs.

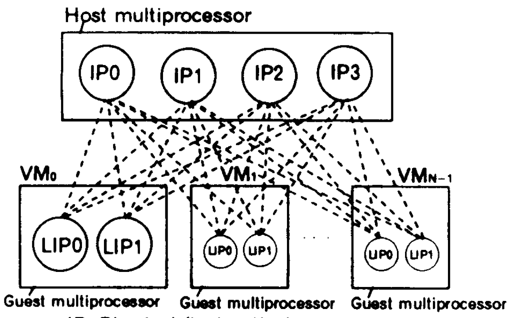


Fig. 2 Floating scheduling.

3. Problem in Managing TLB Entries for VMs

The CPU performance depends on the not-in-TLB-ratio (NITR), which means the ratio of the number of translations outside the TLBs to the number of instructions executed. The NITR has to be low in a VM environment, as in a real machine, but tends to be higher than in a real machine. We describe the main causes for this.

3.1 Conventional Methods

3.1.1 VM Identifiers in TLBs

In the early days of computing, VMs were implemented in a uniprocessor-mode host, which had only one real processor. This real processor had TLBs, which were designed only for a native-machine-mode environment. The VMM had to clear TLB entries for a running VM whenever it switched the real processor from the running VM to another VM[1]. For this reason, the NITR was increased for VMs, and the VM performance deteriorated.

Today, VMs are implemented in a multiprocessor-mode host, which has multiple real processors that share main memory. Each real processor supports VM features[2),3),9]. For example, it has VM identifiers or their equivalents in its TLB entries, to recognize TLB entries for multiple VMs. When it interrupts or intercepts a running VM, it can keep all the TLB entries for the VM. Thus, the TLBs can contain entries for multiple VMs. That is, the VMM need not always clear the current real processor's TLB entries for a running VM when it dispatches another VM on the same real processor.

3.1.2 TLB Control for Multiprocessors

A real multiprocessor system consists of multiple real processors that share main memory. Each processor has its own translation lookaside buffers (TLBs), which include entries for all the logical addresses that it has ever referred to. A VM can run on any real processors shared by VMs. Therefore, TLB entries for the VM may be contained in any TLB entries of any real processors on which the VM has ever been run. When an OS in a VM that is run in a real processor issues some control instructions that purge some TLB entries, the real processor has to purge its TLB entries associated with the VM because the VM should not use them.

A guest multiprocessor system, which means a multiprocessor system of a VM, consists of several logical processors that share their main memory area. Such systems are widely used to enhance system performance. A real processor may have TLB entries containing logical processor identifiers that allow the TLBs to contain entries for multiple logical processors. In this case, the real processor uses the identifiers in its TLBs to determine whether or not the TLBs contain logical addresses that the currently running logical processor has referred to.

TLB entries are purged by purge-TLB-type instructions, which consist of non-signalling-purge-TLB-type (NPTLB) instructions and signalling-purge-TLB-type (SPTLB) instructions. An NPTLB instruction purges TLB entries only from the current processor. On the other hand a SPTLB instruction purges TLB entries from the current processor and signals all other processors configured on-line to purge their own TLB entries. That is, it synchronizes the purging of TLB entries from all processors configured on-line.

An OS issues purge-TLB-type instructions in the following cases:

1. When it releases its real memory area for jobs that have just ended or have just been swapped out. The memory area is used to run a new job. The OS has to change the mapping from its virtual memory area to its real memory area, because the new job to be run requires a new memory mapping. For this purpose, the OS issues SPTLB instructions or NPTLB instructions to purge old TLB entries that contain any logical addresses the ended or swapped-out jobs have ever referred to. In this case, the OS releases all the real pages of the jobs.
2. When it releases real pages in its paging activities. In this case, the OS releases only one real page when stealing a real page of a process and allocating it to another pro-

cess.

The frequency with which these instructions are used is not very high, because these events do not occur frequently. It depends on the hardware functions and design of an OS whether the OS uses NPTLB instructions or SPTLB instructions. Both types increase the NITR in a real machine environment. In a VM environment they cause an additional increase of the NITR, as we will explain in the following section.

3.2 Problem

The problem we have to solve is the additional increase of the NITR in VM mode. The NITR in a VM environment tends to be higher than that in a real machine environment. This is because a real processor tends to purge TLB entries excessively in a VM environment in order to simplify the method for processing purge-TLB-type instructions, which consist of NPTLB instructions and SPTLB instructions.

(1) SPTLB instruction

An SPTLB instruction has several native parameters, such as a page index of a virtual page address, and a real page address to be invalidated.

In native mode, a real processor purges TLB entries that are selected strictly according to the native parameters. The current real processor transmits all the native parameters to all the other real processors that are configured, and signals them to purge their own TLB entries selected according to the parameters. Therefore, no excessive purging occurs.

In VM mode, a real processor has an additional parameter, which is the identifier of a logical processor that belongs to a VM and has issued purge-TLB-type instructions. For SPTLB instructions, a real processor should purge the TLB entries for the logical processor and all other logical processors belonging to the same VM. The TLB entries to be purged should be selected strictly according to the native parameters. Moreover, the real processor has to signal all other real processors to purge their TLB entries for all the logical processors of the same VM. The entries to be purged should also be strictly selected according to the native parameters.

These processes in VM mode require long and complex microprogram coding. To simplify them, a real processor tends to ignore some parameters of the instructions, and consequently tends to purge its TLB entries excessively. One of the following simplifications is used:

1. A real processor ignores the identifier of a logical processor, and purges TLB entries irrespective of the identifiers of logical processors in its TLBs. In this case, it purges TLB entries not only for the VM whose logical processor has just issued the SPTLB instruction, but also for all the other VMs.
2. A real processor ignores the native parameters, and purges all the TLB entries associated with all the logical processors of the VM. In this case, it purges even valid TLB entries for the VM.

An OS uses SPTLB instructions mostly for its paging and swapping activities. When it has enough memory for its workloads, the OS does not so often use the SPTLB instructions. For this reason, under these conditions, the frequency of the SPTLB instructions used is low, and such excessive purging of TLBs has little influence on the increase in the NITR. These types of simplification are therefore reasonable, because they reduce the amount of microprogram coding for the instructions.

(2) NPTLB instruction

In NPTLB instructions, on the other hand, we find a problem that has more effect on the increase of the NITR than any of the problems in SPTLB instructions. It is explained as follows.

In native mode, an NPTLB instruction purges TLB entries only from the current processor.

In VM mode, an NPTLB instruction purges all the TLB entries associated only with the logical processor that has issued the instruction. The TLB entries may be contained in any TLBs of any real processors on which the logical processor has ever been run. Therefore, the TLB entries of these real processors have to be purged before the real processors run the logical processor. The current real processor does not signal other real processors to purge their TLB entries, because the signalling causes an additional overhead, the NPTLB instruction does not signal other processors to synchronize the purging of TLBs of other processors in its specification, and not all other real processors contain TLB entries for the logical processor that has issued the NPTLB instruction.

To implement an NPTLB instruction in VM mode, a real processor conventionally uses the following method[9]:

> When dispatching a logical processor, a real processor validates and uses its TLB entries for the logical processor only when the real processor is the same as one that last dispatched the logical processor. In other words, this method always purges TLB entries for a logical processor when the running of the logical processor is switched to another real processor.

The details of this method are as follows. The current real processor (IP0) on which a logical processor is being run and has just issued an NPTLB instruction purges its TLB entries for the logical processor. Other real processors may contain their own TLB entries for the logical processor if they have ever run the logical processor. The TLB entries of the other real processors become invalid when the logical processor has issued an NPTLB instruction, and they subsequently remain invalid. Therefore, they have to be purged before the other real processors dispatch the logical processor.

Suppose that one (IP1) of the other real processors is going to dispatch the logical processor immediately after the running of that logical processor has been intercepted or interrupted on a real processor (IP0). The dispatching real processor (IP1) is different from the real processor (IP0) that last dispatched the logical processor. In this case, the real processor (IP0) is called the last host processor of the logical processor.

Conventionally, a logical processor does not remember whether or not it ever issued an NPTLB instruction, and only remembers an address of its last host processor, which last dispatched that logical processor. When dispatching a logical processor, a real processor checks the last host processor address of the logical processor, and if that address is different from its own address, it purges its TLB entries for the logical processor.

This conventional method satisfies the specification of an NPTLB instruction, because TLB entries for a logical processor that has issued the instruction on the current real processor are purged on the current real processor at the same time as the instruction is issued. Moreover, the TLB entries are purged immediately before the logical processor is dispatched on any real processors other than the current real processor. This method excessively purges TLB entries for a logical processor, because it always purges them whenever the logical processor is switched to another real processor, whether or not the logical processor has ever issued NPTLB instructions. This method, called "the last-host-CPU-address" method[9], is reasonable if the VMM tries to schedule a logical processor on the same real processor as far as possible. In particular, it has no problem with the fixed scheduling, because there is no processor switching, as shown in Fig. 1. An OS uses NPTLB instructions mostly when its job has ended or been swapped out. Therefore, the frequency with which NPTLB instructions are used is low. The frequency of the excessive purging in the conventional method depends on the frequency with which logical processors are switched to other real processors configured online.

There is a problem with the above method in the floating scheduling shown in Fig. 2. This problem greatly increases the NITR for some workloads. In the conventional method, TLB entries for a logical processor are purged whenever the logical processor is switched to another real processor. In floating scheduling, any free real processor can run any ready logical processor. When the VMM schedules a logical processor, the last host processor of the logical processor may be busy scheduling another logical processor. In that case, another free real processor different from the last host processor has to schedule the logical processor. Thus, a logical processor may be switched to another real processor that runs it, however hard the VMM tries to schedule a logical processor on the same real processor. Moreover, for some workloads, logical processors are frequently switched to other real processors by the VMM. Therefore, the frequency of this type of excessive purging is high, and as a result the NITR is increased and the performance deteriorates for some workloads in floating scheduling. Despite this problem, floating scheduling is normally used, because it is more flexible and gives a better performance than fixed scheduling. Therefore, it is important to solve the problem. We describe a method for doing so in Section 4.

3.3 Influence of Not-in-TLB-Ratio on Performance

Programs run with logical addresses, which real processors translate into system real addresses by using TLBs. When they cannot do this with TLBs, real processors use translation tables, which are managed by OSs. They put the logical addresses and corresponding real addresses into their TLBs. This table-look-in translation is 10–50 times slower than translation using TLBs. Therefore, the execution speed of instructions is greatly dependent on the NITR.

Recently, real processors have began to support several features for VMs in their hardware architectures[2),3),9)]. For example, the TLBs contain VM identifiers or their equivalents, VM logical addresses, and corresponding real addresses. Therefore, the VM CPU performance also depends on the NITR. Programs in VMs run with VM logical addresses that real processors translate into system real addresses by using TLBs. When they cannot do this with TLBs, real processors use translation tables, which are managed by guest OSs or the VMM. They also put the VM logical addresses, corresponding real addresses, and VM identifiers into their TLBs.

4. New Method of Managing TLB Entries for Logical Processors

4.1 Method

The TLB entries for a logical processor become invalid only when the logical processor has issued purge-TLB-type instructions, which consist of SPTLB instructions and NPTLB instructions. A real processor should invalidate the TLB entries for a logical processor that has ever run on it only if that logical processor has ever issued purge-TLB-type instructions. Since the frequency of purge-TLB-type instructions is low, we can reduce the frequency with which TLB entries for logical processors are purged, by restricting the purging of TLBs to cases in which purge-TLB-type instructions have been issued.

SPTLB instructions signal all real processors to purge their TLB entries, and the purging is synchronized with the processing of the instructions. The frequency of purging due to such instructions is low, and their influence on the increase in the NITR is also low, as explained in Section 3.2.

NPTLB instructions, on the other hand, purge the current real processor's TLB entries, and defer the purging of any other real processor's TLB entries until that real processor actually runs the logical processor that has just issued NPTLB instructions. To avoid excessive purging of TLB entries, a real processor uses a special means to remember that a logical processor has issued an NPTLB instruction. When dispatching a logical processor, the real processor checks the means and determines whether or not the logical processor has ever issued NPTLB instructions. TLB entries for that logical processor become invalid only if it has ever issued NPTLB instructions. Only in this case, therefore, real processors invalidate their TLB entries for the logical processor before they dispatch it. Otherwise, they reuse their TLB entries for the logical processor even if the logical processor has been switched. We now describe the method.

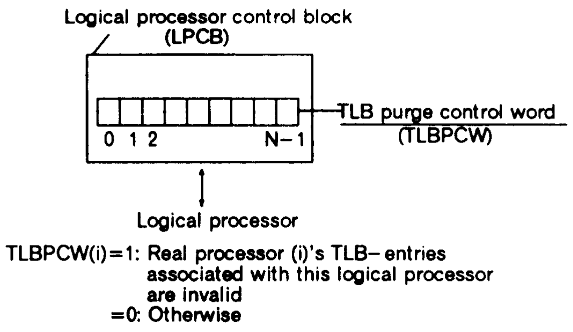

Logical processor control block (LPCB)
0 1 2 N−1
TLB purge control word (TLBPCW)
Logical processor
TLBPCW(i)=1: Real processor (i)'s TLB−entries associated with this logical processor are invalid
=0: Otherwise

Fig. 3 TLB purge control word (TLBPCW) of logical processor.

We present a means whereby real processors can check the validity of TLB entries for logical processors when the VMM has switched the running of logical processors to other real processors. The means contains a new control word, called "TLB-purge-control-word (TLBPCW)," to represent the validity of TLB entries for a logical processor. This word enables a logical processor to remember that it has issued an NPTLB instruction. A control word is defined for each logical processor, as shown in **Fig. 3**. The control word consists of N bits for an N-way host multiprocessor. When the mask (i) of the control word of a logical processor equals 1, it means that real processor (i)'s TLB entries associated with the logical processor are invalid. The VMM manages the TLBPCW as shown in **Fig. 4**:

(1) The initial value of the TLBPCW is 0.

(2) When a logical processor (LIP0 in Fig. 4) is run by a real processor (i) (IP0 in Fig. 4, ①) and has issued an NPTLB instruction (②), all the entries except for the i-th entry of the TLBPCW of the logical processor are set to 1. This means that the logical processor has issued NPTLB instructions, so that, for

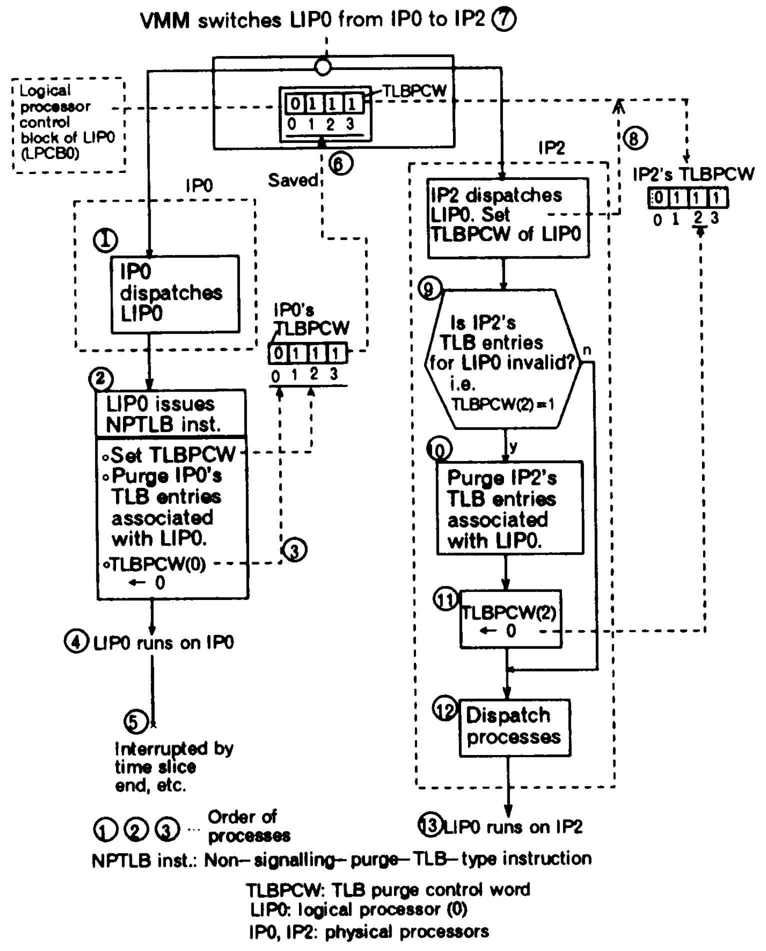

VMM switches LIP0 from IP0 to IP2 ⑦
Logical processor control block of LIP0 (LPCB0)
TLBPCW
0 1 1 1
0 1 2 3
Saved ⑥
IP0
① IP0 dispatches LIP0
IP0's TLBPCW
0 1 1 1
0 1 2 3
② LIP0 issues NPTLB inst.
∘Set TLBPCW
∘Purge IP0's TLB entries associated with LIP0.
∘TLBPCW(0) ← 0
③
④ LIP0 runs on IP0
⑤ Interrupted by time slice end, etc.
IP2
⑧
IP2's TLBPCW
0 1 1 1
0 1 2 3
IP2 dispatches LIP0. Set TLBPCW of LIP0
⑨ Is IP2's TLB entries for LIP0 invalid? i.e. TLBPCW(2)=1
n
y
⑩ Purge IP2's TLB entries associated with LIP0.
⑪ TLBPCW(2) ← 0
⑫ Dispatch processes
⑬ LIP0 runs on IP2
①②③ ··· Order of processes
NPTLB inst.: Non-signalling-purge-TLB-type instruction
TLBPCW: TLB purge control word
LIP0: logical processor (0)
IP0, IP2: physical processors

Fig. 4 Remembering issuance of NPTLB instructions.

any j (≠i), real processor (j)'s TLB entries for the logical processor are invalid. Real processor (i) purges its TLB entries associated with the logical processor. It also clears the i-th entry of the TLBPCW of the logical processor (③).

(3) Real processor (i) (IP0 in Fig. 4) runs the logical processor (LIP0 in Fig. 4) (④). Its running is temporarily halted by several interrupts, such as a time slice end (⑤), and IP0 stores its TLBPCW in the logical processor control block of LIP0 (LPCB0 in Fig. 4) (⑥). Control is then transferred to the VMM.

(4) The VMM switches the logical processor (LIP0) from real processor (i) (IP0) to real processor (j) (IP2) (⑦).

(5) When it has dispatched a logical processor (LIP0 in Fig. 4), real processor (j) (IP2 in Fig. 4) sets up the TLBPCW of the logical processor in its internal storage (⑧) and checks the mask of the TLBPCW (⑨). If the j-th entry of the TLBPCW is 1, it means that real processor (j)'s TLB entries associated with the logical processor are invalid. Therefore, the real processor purges those TLB entries (⑩), and clears the j-th entry of the TLBPCW to 0 (⑪).

(6) Real processor (j) (IP2) continues dispatching processes (⑫), and runs the logical processor (LIP0) (⑬).

This method does not excessively purge TLB entries, because the TLB entries associated with a logical processor are purged only if that logical processor has ever issued NPTLB instructions.

In other implementation, the VMM creates and manages the above-described TLBPCWs. In this case, the real processor running a logical processor has to notify the VMM that the logical processor has issued an NPTLB instruction. It can notify the VMM by setting a flag, which indicates the issuance, in the LPCB of the logical processor that issued the NPTLB instruction. The VMM then has to make a TLBPCW in the LPCB of the logical processor. A real processor has to check and update the TLBPCW of a logical processor that it has dispatched in the same way as shown in ⑨, ⑩, and ⑪ of Fig. 4.

4.2 NITR Evaluation

(1) Evaluation from processor scheduling

Consideration:

In the fixed scheduling shown in Fig. 1, the VMM does not switch the logical processors to other real processors. Therefore, the conventional method does not cause excessive purging of TLBs due to processor switching, because there is no processor switching, and thus our new method is not effective for this type of scheduling.

In floating scheduling shown in Fig. 2, on the other hand, the conventional method may excessively purge TLB entries when logical processors are switched to other real processors. Our method purges TLB entries for logical processors only when the entries are invalid owing to the logical processors' issuance of NPTLB instructions. Therefore, it does not excessively purge TLB entries as a result of the switching of logical processors. Thus, it is effective for this type of scheduling.

In fixed scheduling, no logical processor is switched between real processors, and TLB entries for a logical processor are not purged unless it issues purge-TLB-type instructions. The proposed method does not purge TLB entries for a logical processor unless it issues purge-TLB-type instructions even if it is switched to another real processor. Thus, in terms of TLB-purge control, the proposed method corresponds to fixed scheduling in conventional TLB maintenance. Therefore, we can estimate the effect of the proposed method by comparing the NITR of fixed scheduling with that of floating scheduling in conventional TLB maintenance.

The proposed method enables a real processor to reuse TLB entries for a logical processor when that logical processor is switched to it. The method is effective only in conditions where logical processors are frequently switched between real processors. Therefore, we evaluate the method under those conditions.

Measurement data:

In conventional TLB control, **Table 1** shows measurement data for a workload of heavy I/O jobs with CPU utilization of about 30% per real processor, in a two-way host multiprocessor, with two VMs, each in two-way mode. A processor has buffer storage, which is tens of times faster and thousands of times smaller than main storage and contains copies of the data in main storage. The not-in-buffer ratio (NIBR) is the ratio of the number of memory accesses to locations not in the buffer storage to the number of instructions executed. The CPU performance is heavily dependent on the NIBR, to an extent of the same order as its dependence on the NITR.

We used a two-way real multiprocessor and two VMs that are themselves two-way multiprocessors. Workloads contain heavy I/O jobs, and the CPU utilization of each real processor is low. Therefore, the logical processors of those VMs can be switched frequently between real processors.

The proposed method enables a real processor to reuse TLB entries for a logical processor when the logical processor is switched to it. The method is effective only in conditions where logical processors are frequently switched between real processors. Therefore, we evaluated the method under those conditions. As we explained, we can estimate the effect of the proposed method by comparing the NITR of fixed scheduling with that of floating scheduling in conventional TLB maintenance.

Table 1 shows that the NIBRs are of the same

Table 1 Influence of scheduling on NITR and NIBR.

#	Items	Fixed scheduling	Floating scheduling
1	VMEI Msteps	3,809.5	3,718.0
2	VM NITLB (freq. M times)	21.9	80.5
3	**VM NITR**	**0.6% (1)**	**2.2% (3.78)**
4	VM NIB (freq. M times)	223.8	252.9
5	**VM NIBR**	**5.9% (1)**	**6.8% (1.2)**

Workload:
- 2-way host multiprocessor
- 2 VMs, each 2-way
- Heavy I/O jobs
- CPU utilization: 30% per real processor
- #1–#5: per real processor

Comments:
- VMEI: Instruction execution steps in VM mode
- VM NITLB: Not in TLB in VM mode
- VM NITR: Not-in-TLB ratio in VM mode
- VM NIB: Not in buffer in VM mode
- VM NIBR: Not-in-buffer ratio in VM mode

order. On the other hand, the NITR in floating scheduling is about four times higher than that in fixed scheduling. These data indicate that, in floating scheduling, logical processors are frequently switched between real processors and their TLB entries for logical processors are excessively purged in the switching in conventional TLB control. Our method does not purge TLB entries excessively, but purge them only when they are invalid. Consequently, it can reduce the NITR in floating scheduling to the same level as in fixed scheduling.

(2) Evaluation from a simulation in a uniprocessor

The method described in Section 4.1 can avoid excessive purging of TLBs when logical processors are switched by the VMM. This is because it enables real processors to purge TLB entries associated with a logical processor only when that logical processor has issued purge-TLB-type instructions.

To evaluate the method, we simulate the worst case by using a uniprocessor host and three VMs, each in uniprocessor mode. Suppose that the VMM always switches logical processors to other real processors when dispatching the logical processors. In this case, conventionally, at every dispatch of a logical processor on a real processor, the real processor purges the TLB entries associated with the logical processor. On the other hand, our method keeps almost all the TLB entries associated with the logical processor at every dispatch, because jobs running in the logical processor issue few purge-TLB-type instructions while they are running. We can obtain the NITR in the two methods (the conventional method and the new method) as follows. The mainframes used for measurement have two internal hardware options for TLB maintenance:

(1) Option 1 (Keeping method): The hardware or microprogram keeps the TLB entries for logical processors at every exit from VM mode to VMM mode.

(2) Option 2 (Clearing method): The hardware or microprogram purges the TLB entries for logical processors at every exit from VM mode to VMM mode.

The conventional method corresponds to the method used in option 2 in the worst case. Our method corresponds to the method used in option 1 as explained above. Three VMs are defined and operated concurrently for this measurement. They are all in uniprocessor mode,

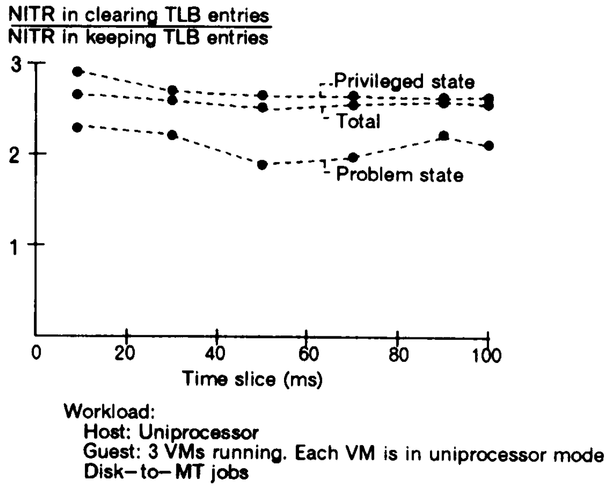


Fig. 5 NITR increase caused by clearing TLB entries at every exit from VM mode to VMM mode.

and use I/O direct execution[3),7)]. Jobs consisting of dumping from disks to MTs are used as the workload, to maintain a high frequency of I/O instruction issuance.

We used one real uniprocessor and three VMs that were all uniprocessors. Since the workload consists of disk-to-MT dumping jobs, the real processor can be switched frequently between VMs. Since only one real processor is used, the VMs are always run on it. Moreover, they are frequently interrupted by I/O wait, I/O interrupt, and time-slice-end, and they frequently exit from VM mode to native mode. The TLB entries for the VMs are cleared or kept at the every exit, depending on the above-mentioned hardware options. Therefore, by using these hardware options and conditions in which VMs are frequently interrupted, we can simulate the proposed method and estimate its effect.

The results are shown in **Fig. 5**. The NITR of option 2 is more than twice as large as that of option 1. That is to say, the NITR of our method, which is called the "remembering NPTLB-issuance" method, is less than half the NITR of the conventional "last-host-CPU-address" method.

4.3 Performance Considerations

The not-in-buffer ratio (NIBR) and the not-in-TLB ratio (NITR) strongly influence the mean-instruction-execution time (MIET). The VMM uses only a few TLB entries, because the locality of its memory references is high and its overhead is low[2),3)]. Even if the TLB entries of the VMM have the same columns in the TLBs as the TLB entries of a running VM, they will be replaced by those of the VM according to the least recently used (LRU) algorithm, because the execution steps of the VM are much larger than those of the VMM. Therefore, the TLB entries of the VMM have little influence on the TLB capacity for VMs.

We estimate the influence of the NITR on the MIET in a VM environment. Our proposed method does not address the NIBR. The NIBR in our method and that in the conventional method are of the same order, as shown in Table 1. We ignore the difference between the NIBRs of VMs. The MIET of VM1, that is, *MIET1*, is expressed as follows:

$$MIET1 = T1 + NITR1^{*} AT1,$$

where

T1: Basic instruction execution time of VM1

NITR1: NITR of VM1

AT1: Address translation time of VM1 when address translation tables are used.

The MIET of VM2 is expressed similarly. Since real host processors support VM architecture,

AT1 and *T1* are nearly equal to *AT2* (= *AT*) and *T2* (= *T0*), respectively.

Suppose that *NITR2* is smaller than *NITR1*. Then, *MIET2* is smaller than *MIET1*. The MIET reduction ratio (*D*) of VM2 to VM1 is expressed as follows.

$$D = \frac{MIET1 - MIET2}{MIET1} = \frac{(NITRI1 - NITR2)^{*} AT}{MIET1}.$$

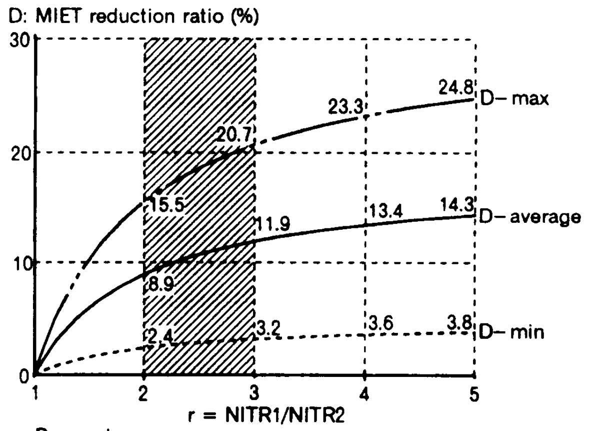
Therefore,

$$D = \frac{(1 - \frac{1}{r})}{1 + \frac{T0}{NITR1^{*} AT}},$$

where,

$$r = \frac{NITR1}{NITR2}.$$

In usual mainframes the basic instruction execution time of a real processor is 2–4 machine cycles (mc). Because the real processor supports the VM architecture, the basic instruction execution time of a VM is almost the same as that of the real processor. Therefore, *T0* = 2–4 mc. The address translation time (*AT*) is usually 20–30 mc. We presume that *NITR1* = 1–3%, which is conventional and a little higher than that of a real machine. From these parameters, we get that *MIET1* = 2.2–4.9 mc. The relation between *r* and *D* is shown in **Fig. 6**.



Parameters:
NITR: Not-In-TLB ratio; NITR1: NITR of conventional VM (VM1)
NITR2: NITR of VM2 with new TLB maintenance
MIET: Mean instruction execution time; MIET1: MIET of VM1
MIET2: MIET of VM2
D = (MIET1-MIET2)/MIET1
Assume that
NITR1 = 1% - 3%; NITR2 < NITR1; i.e., r > 1

Fig. 6 MIET reduction ratio (*D*).

The figure shows that the CPU time of VM2 is 9–12% less than that of VM1 when *NITR2* = (1/2 to 1/3)**NITR1*. This *NITR2* is estimated from the effect of the proposed method.

5. Conclusion

Real processors translate VM logical addresses into system real addresses by using translation lookaside buffers (TLBs). VM CPU performance is greatly dependent on the not-in-TLB ratio (NITR), which is the ratio of the number of address translations outside the TLBs to the number of instructions executed. The VMM or hardware/micro-programs tend to excessively purge TLB entries for VMs, in order to simplify simulation processes for VMs. Therefore, the NITR in VMs tends to be higher than in real machines.

Conventionally, a real processor purges its TLB entries for a logical processor of a VM whenever the real processor is different from the last host processor, which is a real processor that last dispatched the logical processor, whether or not the logical processor has ever issued a non-signalling-purge-TLB-type instruction, which purges TLB entries only from the current processor. This causes excessive purging of TLB entries and increases the NITR of VMs, consequently reducing the VM CPU performance.

We have presented a method for remembering that a logical processor has ever issued non-signalling-purge-TLB-type instructions, which purge TLB entries only for the logical processor, and for making the real processor that is about to run a logical processor reuse its TLB entries for that logical processor, even when the real processor is different from the last host processor of that logical processor, if the logical processor has not ever issued such instructions since the last time they were purged. The real processor purges its TLB entries for the logical processor only if the logical processor has issued such instructions.

This method can avoid the conventional excessive purging of TLB entries for logical processors, and reduces the NITR of VMs to 1/2–1/3 that of VMs in the conventional method, thus reducing the VM CPU time by 9–12% of that in the conventional method.

Acknowledgment

We express our sincere thanks to T. Onose, K. Une of General Purpose Computer Division, Hitachi, Ltd., T. Inoue, S. Tanaka of Systems Development Laboratory, Hitachi, Ltd., and Mr. R.E. Abeles of Hitachi Data Systems, Ltd., for their performance measurement and their technical comments.

We also express our sincere thanks to T. Kuwabara, Y. Ohshima of General Purpose Computer Division, Hitachi, Ltd., for their helpful management.

References

1) Goldberg, R.P.: Architectural Principles for Virtual Computer Systems, Ph.D. Dissertation, Div. Eng. Appl. Phys., Harvard Univ., Cambridge, MA (1972).
2) Borden, T.L., et al.: Multiple Operating Systems on One Processor Complex, *IBM Systems J.*, Vol.28, No.1, pp.104–123 (1989).
3) Umeno, H. and Tanaka, S.: New Methods for Realizing Plural Near-Native Performance Virtual Machines, *IEEE Trans. Comput.*, Vol.C-36, No.9, (Sept. 1987).
4) Ono, K.: A Method for Improving Virtual Machine System Performance, Fujitsu, OS Working Group of Information Processing Society of Japan (WGOS-IPSJ), No.18, (Feb. 4, 1983).
5) Ebino, Y., et al.: Extended Virtual Machine System for ACOS System 2000 Series, *NEC Technical Journal*, Vol.40, No.11 (1987).
6) Doran, R.W., et al.: Amdahl Multiple-Domain Architecture, *Computer*, pp.20–28 (Oct. 1988).
7) Bean, G.H., et al.: Logical Resource Partitioning of a Data Processing System, IBM, Priority, July 29, 1987 (PR/SMTM Patent).
8) Umeno, H. and Ohmachi, K.: A Method for Supporting Virtual Machine Multiprocessor System, *Proc. of the 19th Annual Convention IPS Japan*, pp.265–266 (1978).
9) IBM: IBM System/370-XA Start Interpretive Execution, SA22-7095.
10) Umeno, H., et al.: Performance Improvement Methods for Virtual Machine System, *J. IPS Japan*, Vol.31, No.12, pp.1665–1680 (Dec. 1990).
11) IBM: Enterprise Systems Architecture/390 Principles of Operation, SA22-7201-01.

(Received April 19, 1994)
(Accepted November 2, 1995)

Hidenori Umeno was born in Ohita, in 1947. He received the B.S. in mathematics from Kyushu University in 1970. From 1970 to 1976, he was with Central Research Laboratory, Hitachi, Ltd., where he made researches in productivity improvement of compilers. From 1976 to 1993, he was with Systems Development Laboratory, Hitachi, Ltd., where he made researches in performance and reliability improvement of virtual machines, file systems, and operating systems. Since 1993, he has been with General Purpose Computer Division, Hitachi, Ltd., where he has been engaged in the development of logical partition systems of mainframes. His main research fields are performance and function improvement of virtual machines, logical partition systems, operating systems, and computer architectures. He received a best paper award of Information Processing Society of Japan (IPSJ) in 1982. Since 1991, he has been a part-time instructor of Musashi Institute of Technology. He is a member of IPSJ and ACM, and an affiliate of IEEE Computer Society.

Hiroshi Ikegaya was born in 1957. He entered Systems Development Laboratory, Hitachi, Ltd., in 1975, in which he had been engaged in research on mainframe architectures and virtual machine systems until 1992. From 1992 to 1995, he was with Development Department I of General Purpose Computer Division (GPC), Hitachi, Ltd., and engaged in the design and development of large mainframes. Since 1995, he has been with Information Systems Development Center of GPC, and engaged in the development of information management systems in basic business. He is a member of IPSJ, and interested in the development of business process reengineering and client server systems.