Virtual Address Translation for Wide-Address Architectures*

Ing-Jye Shyu and Shiuh-Pyng Shieh
Department of Computer Science and Information Engineering
National Chiao Tung University, Hsinchu, Taiwan 30050, R.O.C.
E-mail: ssp@csie.nctu.edu.tw

Abstract

Operating systems employ virtual memory mechanism to provide large address space for programs. The efficiency of the virtual address translation plays an important role in determining system performance. In conventional virtual memory management systems, both the forward-mapped page table scheme and inverted page table scheme are widely used to organize the page tables that record translation data. These two schemes work well for 32-bit architectures, but not for wide address (64-bit) architectures. These two schemes either needs long translation time or incurs large memory overhead when translating wide virtual address. To cope with this problem, we propose a hybrid scheme to accelerate wide virtual address translation.

1. Introduction

Due to the recent debut of 64-bit powerful microprocessors such as Alpha by DEC, PA-RISC by HP, Pentium by Intel and R4400 by MIPS, the design philosophy of operating systems, such as UNIX, need be reconsidered with respect to these new architectures, The key distinctions between 32-bit and 64-bit architectures are the addressibility of virtual memory space and the data transferring bandwidth. In this paper, we propose a new scheme that can effectively translate the huge 64-bit virtual address.

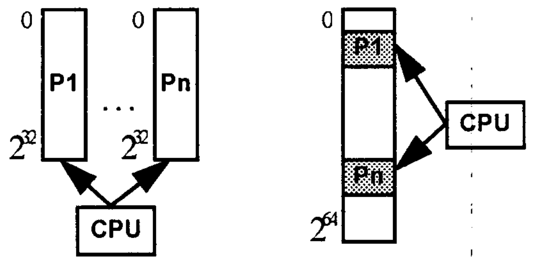
In general, virtual memory is a fundamental abstraction of storage used by computer systems to support concurrent execution of processes. Virtual memory management defines the mapping function from virtual address space to physical address space. The processor interrogates the TLB (Translation Lookaside Buffer) with a virtual address and searches for the corresponding translation.

* This work was supported by the National Science Council, Taiwan, R.O.C., under the contract NSC84-2221-E-009-001

If found, called a hit, the hardware retrieves the related protection information and physical page data. If the TLB does not contain the data, called a miss, the hardware will search a memory-based table, the page table, for a match and then store this translation relation into the TLB to improve performance. Thus, the system performance is dominated by the TLB hit ratio and the virtual memory management overhead.

The process context switch substantially affects the TLB hit ratio. Most of the current operating systems use the 32-bit virtual address for a process, and each virtual address space is a separate protection space. With hardware support, a 32-bit virtual address space can be accessed only by its owner process. Figure 1(a) shows the conventional virtual memory management structure in which every process has the same 32-bit virtual address space, and only a set of mapping information belonging to current executing process is active in a time slice. When context switch occurs, the system must flush the TLB content and reset the virtual memory mapping information. The TLB flushing operation incurs a large number of TLB misses and thus degrades system performance.

Conventional virtual memory management need be redesigned in order to fully utilize the new processors that support 64-bit virtual addressing capability. A process, running on a 64-bit processor, will not be restricted within an isolated 32-bit virtual address space. Instead, all processes coexist in the huge 2^{64}-byte virtual space, which is divided into 2^{32} equal-size partitions. Every process resides in a partition and has a different 64-bit virtual address range, see Figure 1(b). The left-most 32 bits and the right-most 32 bits contain the process ID and the address space of this process, respectively.



(a)32-bit virtual address space. (b) 64-bit virtual address space.

Figure1

Since processes have different address ranges and locate at the same 2^{64} virtual address space, the context switch only need an address change from the range of an old process to that of a new process. The mapping relationship defined by the TLB need not be flushed, thereby reducing the TLB misses and the context switch cost. The 64-bit architecture also affect the design of other parts of operating systems. For example, the entire file system can be mapped to this huge 2^{64} virtual space (the 2^{64} space is so huge that you could use 100 MB per second for five thousand years); the invocation of a RPC at local site in 64-bit architectures is almost the same as that of a local procedure call; 64-bit address space can provide real network transparency in a local area network. Despite of all the merits the 64-bit architecture has, the technique for efficient virtual address translation remains unexplored.

Conventionally, the most popular translation schemes can be classified as the forward-mapped page table translation and the inverted page table translation. Both work well in 32-bit memory architectures. However, they cannot fully utilize the merits of 64-bit memory architectures. The technique used by current 64-bit microprocessors is so-called software managing TLB. If a miss occurs, the TLB invokes a TLB fault and then transfers the control to a TLB fault handler which is a part of operating systems. The handler searches a virtual memory maintaining table for address translation and then inserts it to TLB. This approach hides the hardware complexity for maintaining a page table but increases the translation time for software manipulation.

In the next section, we will introduce the forward-mapped page table translation and the inverted page table translation, as well as their drawbacks when being applied to 64-bit translation. In section 3, we will propose a hybrid translation scheme to overcome these drawbacks. In section 4, We will evaluate these three schemes with metrics by comparing translation performance and memory-overhead. Finally, we give a conclusion in section 5.

2. Virtual Memory Architecture

Two types of virtual-to-physical address translation architectures are widely used nowadays: *forward-mapped page tables* and *inverted page table* schemes. However, they suffer from the drawbacks of performance degradation and memory-consuming when applying to the virtual address translation of 64 bits or higher.

2.1 Forward-mapped page tables

The forward-mapped page table scheme uses part of the virtual address as indexes to a hierarchical page table. Figure 2 shows an example illustrating conventional 32-bit virtual-to-physical address translation. The base address register is an internal register which points to the first level of the page table. The complete physical address is composed of both the offset bits in the virtual address and the physical page number derived from the second level of the page table. In order to avoid address translation for every memory access, the mapping information for translation from virtual page numbers to physical page numbers is cached in the TLB. The forward-mapped page table is a per-process table. When a context switch occurs, the base address register will be changed to pointing to the page table of the new process. The size of physical memory used for the page table is a function of the amount of the virtual memory allocated by the current process. The page of a page table need not reside in memory if none of its entries are used. Therefore, we divide the pages of a page table into two partitions: the active pages and the inactive pages. The active pages are stored in memory and the inactive pages are stored in the secondary storage. An inactive page will be swapped into memory when its entries are used for mapping virtual address. When the memory allocated to a page table is exhausted, a resident page will be replaced by the inactive page being requested.

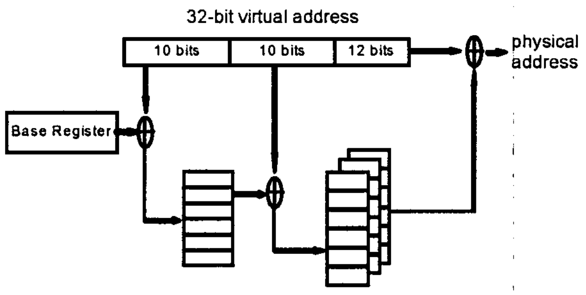


Figure 2: Forward-mapped page tables

Consider the following example. For a process with a code region of 128KB, a static data region of 128KB and a stack region of 8KB in 32-bit architectures, it requires a table of at least 4 pages for translation, with the assumption that data access is sequential in each region. In this case, the overhead is 4*4K/264K=6%.

To translate a full 64-bit virtual address, the forward-mapped translation scheme needs to use a five-level page table to translate the 52-bit wide virtual address page number to the physical page number, see Figure 3. The size of all pages of the table is 4K bytes except the root page, which is 16K bytes. In this way, a lot of memory is wasted for the maintenance of this five-level page table. As the addressing bits grows, the number of table levels needs to grow or the page size needs to increase. For the same example described above, the memory-consuming overhead is at least (12*4K+16K)/264K=24%, and the time to translate virtual address is also lengthened. This overhead suggests the consideration of alternative structures and approaches.

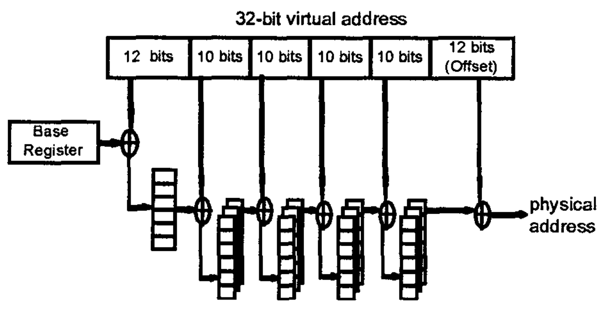


Figure 3: A five-level page table.

2.2 Inverted Page Tables

Conventionally, the implementation of large address space prefers using *inverted page tables*. The inverted page table is a global system table, where each entry records the protection bits and the mapping from the virtual page number to the physical page number. TLB is also used to boost the translation performance. When a miss occurs, the TLB fault handler first search this inverted page table to determine which physical page currently associates with this virtual page, see figure 4. The searching time is determined by the physical memory size. For example, a computer has 4MB physical memory, in which each physical page size is 4096 bytes and each entry needs 4 bytes to describe the properties of virtual memory page (This specification is common in many personal computer systems). This computer requires 4KB inverted page table and a mean of 4KB/2*4=512 times to linearly search the inverted page table. The substantial search overhead can be reduced by using a hash function. However, the hardware that supports the collision-free hash function is quite complicated.

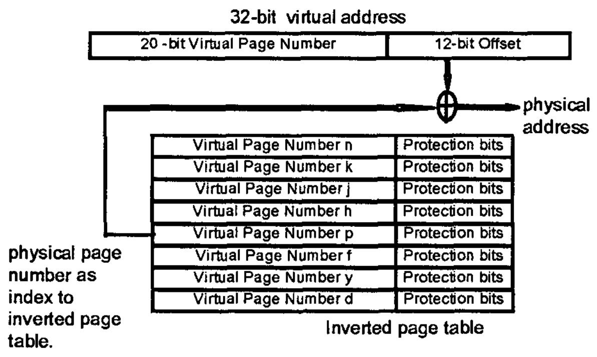


Figure 4: Inverted page table.

Address aliasing is another problem of using inverted page table. Only one virtual page can be mapped to a physical page at any instant, thereby making page sharing among processes difficult. In the inverted page table scheme, an extra management table is needed to maintain the virtual page to physical page mapping relationship. If the search fails, (that is, the intended page is currently swapped out to secondary storage) the management table will provide extra information about where the intended page is. Thus, the intended page can be swapped back in. To record memory management information in both data structures, the inverted page table and the management table, is expensive and inefficient.

3. The Hybrid Translation Scheme

In order to compensate for the inadequacy of current schemes, we propose a *hybrid translation scheme*. In this scheme, all processes execute in the 2^{64}-byte virtual address space. The base address TLB (BATLB), which is stored in associate memory, is used to translate a virtual address to a physical address, see Figure 5. The left portion of the BATLB caches the left-most 32 bits of a 64-bit virtual address, namely, process ID, as a tag. The right portion of BATLB is a physical memory address pointing to a two-level page table of this process.

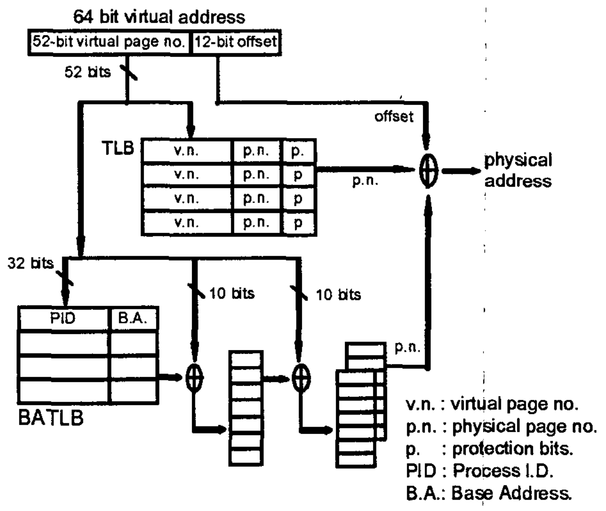


Figure 5: Hybrid translation scheme.

The virtual-to-physical address translation is straight forward. When a 64-bit virtual address is generated, the TLB is searched for a match. If TLB hits, the physical page number returned will be combined with the offset to form a physical address. If TLB misses, that is, there is no virtual-to-physical mapping information in the TLB, the hardware exception occurs and then mapping information must be retrieved by searching a related page table. The left-most 32 bits of the virtual page number is used as a tag to search BATLB for a match. If the BATLB hits, the right portion of the BATLB points to a two-level page table. The hardware automatically retrieves the mapping information, returns it to form a physical address and updates the TLB with this new mapping information. If the BATLB misses, a software exception is invoked and the BATLB fault handler needs to construct the required page table. The BATLB fault handler first searches the internal process tables for a match of process ID. If a match is found, the handler can get the page table from the process table and link it to BATLB. Then the virtual address will be reissued. If no match can be found, the address is illegal and the access will be denied.

The reasons why we use two phase translation (a BATLB cooperates with two-level page tables) are listed as follows:

- If forwarded-mapped page tables are used in 64-bit architectures, the system needs to maintain page tables of five levels. It incurs a large overhead to the system. In general, the left-most 32 bits (that is, process ID) are not fully used, because only a few processes are simultaneously running in a system. Therefore, we introduce the BATLB, which is stored in associated memory, to speed up the address translation. In this way, three levels of pages are saved.

- In a time slice, the virtual address translation usually takes place in the current running process's page table. If five-level page tables are used, the translation will be repeated in the first three-level pages while TLB misses. The BATLB we introduced can completely eliminated the time spent on the repeated translations in the first three-level pages.

On one hand, the hybrid translation scheme uses hardware to accelerate the retrieval of virtual-to-physical mapping information from page tables. On the other hand, it provides the flexibility to modify the BATLB entries to reduce the memory space required to store page tables.

4. Evaluation

For the evaluation, we define some notations to quantify the performance.

$T_{translaton}$	: translation time from a 64-bit virtual address to its physical address.
T_{hit}	: translation time for a TLB hit.
T_{miss}	: translation time for a TLB miss.
$Ratio_{hit}$	: probability of a hit.
$Ratio_{miss}$	: probability of a miss.
t_{ma}	: memory access time.

$M_{physical}$	: size of the physical memory.
M_{page}	: size of a physical page.
N_f	: number of entries in a forward-mapped page table.
N_i	: number of entries in the inverted page table (here is $M_{physical} / M_{page}$).
N_h	: number of entries in each page table used by the hybrid translation scheme.

Assume that the size of a physical/vitual page is 4K bytes, and every page needs a 4-byte entry in forward-mapped page tables or inverted page tables to record translation information. We compare these three schemes by metrics of *translation-time* and *memory-overhead*.

4.1 Translation Time

A 64-bit virtual address consists of two parts: a virtual page number and an offset. When an address translation is invoked, the virtual page number will be searched in TLB for a match. The address translation time can be calculated as follows:

$$T_{translation} = Ratio_{hit} * T_{hit} + Ratio_{miss} * T_{miss}$$

If a TLB hit occurs, the translation time is the same no matter which approaches are used. Therefore, we only focus on the comparison when a TLB miss occurs. For the forward-mapped translation scheme, if a TLB miss occurs, a five-level page table needs to be searched before the physical page number can be determined. If the page table is stored in memory, the translation time for a TLB miss is approximately equal to five memory access time, that is, $T_{miss} = 5 * t_{ma}$. For the inverted page table scheme, an average of $N_i/2 * t_{ma}$ memory access time is needed for a linear search in the inverted page table consisting of N_i entries. In the hybrid translation scheme, the search of TLB and BATLB can parallelly proceed. That is, while the TLB detects a miss, the BATLB also determines the base address pointing to a two-level page table for resolving this miss. Therefore, the time to retrieve the mapping information in our scheme is approximately equal to $2 * t_{ma}$. This analysis suggests that our hybrid translation scheme has the smallest translation time.

This analysis is based on the assumption that there is no secondary page fault when searching the page table. It means that the required five-level page tables are all in memory, the inverted page table contains the intended page and no miss occurs in BATLB.

4.2 Memory Overhead

Among the three schemes, the inverted page table has the least memory-consuming which is a function of the size of the physical memory, that is, N_i * 4 bytes are required. The overhead for maintaining a physical page can be formulated as

$$M_{overhead} = \frac{size\ of\ the\ table\ for\ maintaining\ v-to-p\ translation}{number\ of\ pages\ in\ physical\ memory}$$

Thus, the memory overhead $M_{overhead}$ of using the inverted page table scheme is $N_i * 4/N_i$ = 4. It means that additional 4 bytes are needed to maintain a physical page.

It is difficult to dynamically evaluate the memory overhead when the memory accesses are random. The size of the page table is determined by the distribution of a program's code, data and stack usage. For the same number of accesses, if the accesses to these three regions are sequential, the size of the page table is minimal; if these accesses are randomly distributed in the virtual address space, the size of the page table will increase exponentially.

In order to cope with this problem, we use the static analysis to compare the forward-mapped scheme with the hybrid scheme. Assume that in the worst case only one physical page is accessed, that is, the two-level page table for the hybrid scheme and the five-level page table for the forward-mapped scheme only point to a physical page. The overhead are calculated as $2 \times 4 \times N_h / 1\,page$, for the hybrid scheme, and $5 \times 4 \times N_f / 1\,page$, for the forward-mapped scheme. These two translation schemes incur more memory overhead than the inverted page table scheme. However, the hybrid translation scheme use the same two-level page tables as those used in conventional 32-bit architectures, and incurs the same memory overhead. Therefore, the overhead should be tolerable.

5. Conclusion

The virtual memory translation dominates the performance of an operating system. In this paper, we propose a hybrid translation scheme which is able to efficiently translate 64-bit virtual address. The conventional translation schemes, forward-mapped page table scheme and inverted page table scheme, either need long translation time or incur large memory overhead. Our analysis shows that the proposed hybrid translation scheme provides the best performance for virtual address translation.

References

[1] Albert Chang and Mark F.Mergen," 801 Storage: Architecture and Programming," *ACM transactions on Computer Systems*, pp.25-50, Vol. 6, No. 1, Feb. 1988.

[2] J.S. Chase, " How to use a 64-bit virtual address space," *Technical Report 92-03-0*, Univ. of Washington, Department of Computer Science and Engineering, Mar. 1992.

[3] Digital Equipment Corporation, Maryland, MA., *Alpha Architecture Handbook*, 1992.

[4] Jerry Huck, Jim Hays, " Architectural Support for Translation Table Management in Large Address Space Machines," *20th Annual International Symposium on Computer Architecture ISCA 20*, pp.39-50, 1993.

[5] G. Kane and J. Heinrich, " MIPS RISC Architecture, " *Prentice Hall, Englewook Cliffs, JJ*, 1992.

[6] Eric J. Koldinger, " Architectural Support for Single Address Space Operating Systems, " *In Proc. of the 5th conference on Architectural Support for Programming Language and Operating System*, pp.175-186, 1992.

[7] R.B. Lee. " Precision architecture, " *IEEE Computer*, 22(1), pp.78-91, Jan. 1989.

[8] S.Leffler, M. McKusick, M. Karels, and J. Quarterman, " *The Design and Implementation of the 4.3 BSD UNIX Operating System*, " Addision-Wesley, 1989.

[9] John R. Mashey, " 64-bit Computing," Byte, pp.135-142, Sep. 1991.

[10] Toshio OKAMOTO, Hideo SEGAWA et.al., " A Micro Kernel Architecture for Next Generation Processors," *Proceedings of USENIX 1992 Winter Conference*, pp.83-94, 1992.

[11] Robert M. Supnik, " Digital Alpha Project," *Commuinications of The ACM*, Vol.36, No.2, pp.30-81, Feb. 1993.