A New Page Table for 64-bit Address Spaces

M. Talluri, M. D. Hill, Y. A. Kalidi

Computer Science Department
University of Wisconsin
Madison, WI

Sun Microsystems Laboratories
Mountainview, CA

1. The problem

Programs’ memory usage doubles every one or two years. Most processor architectures are now moving from 32-bit to 64-bit virtual address spaces. Physical address spaces are also increasing although much less dramatically.

How would current page table organizations scale up? We will need to support very large but very sparsely populated address spaces. We also should take into account the fact that many processors (MIPS, Alpha, UltraSPARC) let the kernel handle translation look-aside buffer (TLB) misses. “This makes page table design an operating system issue.”

2. Current Page Table Organizations

There are three main approaches:

1. Linear page tables: since page tables are much too large to fit in main memory, they are to be stored in virtual memory (VMS solution). Supporting very large page tables will require more than 2 levels (3 levels on the MIPS R3000).

2. Forward-mapped page table: the virtual page number (VPN) is divided into fixed-size fields with each field serving as index of a given level of the search tree (SPARC).

3. Hashed (inverted) page tables: page table is organized as a hash table: the VPN is hashed to obtain a bucket address and each bucket is organized as a linked list; hence each page table entry (PTE) contains its VPN, a physical page number (PPN), a few flags and a next pointer (unless the table has fixed-size buckets as in software TLB).

3. Clustered Page Tables

Clustered page tables are a variant of hashed page tables with the difference that each entry stores mapping information for a block of consecutive page table. The number of pages in a page block is called the subblocking factor. Subblocking has several advantages:

1. It requires much less space overhead as we have one VPN and one next pointer per subblock.

2. It takes advantage of the fact that the address space of many programs consists of small clusters of pages that represent individual objects and are scattered anywhere in the address space.

3. It interacts much better with the TLB miss handling firmware or software.

The optimal cluster size is linked to the organization of the TLB.
4. Managing TLB Misses

Superpages and subblocks have been proposed to increase the effectiveness of TLBs.

Superpages are sets of pages that are aligned both in virtual and in physical memory and whose size is a power-of-two multiple of the page size (MIPS, UltraSPARC, Alpha, PowerPC). Large superpages (256KB and above) are especially useful for kernel data.

Subblocking associates multiple physical page numbers (PPN’s) with each TLB tag. With a subblocking factor of 16 and 4 KB pages, each tag covers 64KB of contiguous virtual addresses. Two options can be considered:

1. **Complete-subblocking** allows the page frames containing the pages in a subblock to be placed anywhere in main memory but requires multiple PPN’s for each TLB tag. This is the solution adopted for the MIPS 4X00 with a subblocking factor of 2.

2. **Partial-subblocking** requires the page frames to be placed in a single, aligned block of main memory; there is only one PPN per TLB tag.

The major difference between superpages and subblocks is that all pages in a subblock do not need to be simultaneously present in main memory. Hence a partial-subblock TLB entry must contain a valid bit vector specifying which of the mapped pages are in main memory.

To support superpages, we can:

1. **Replicate the PTEs** and have one copy of the superpage PTE at the page table site of each page in the superpage (simple but does not reduce the size of page tables);

2. Have **multiple page tables** for different superpage sizes (including isolated pages) but each page miss will now have to search several PTs instead of one;

3. Store superpage mappings at the appropriate level of a PT; this only works with multi-level PTs having a **tree structure** and restricts the size of the superpages;

4. Hash on superpage addresses so that all pages that are or could be in the same superpage will be in the same bucket; this only works with hashing and results in larger buckets and longer searches.

Overall, replicating PTE’s seems the best solution. On the other hand, supporting very large superpages could be done on an ad hoc basis as there will be very few of these very large superpages.

Supporting partial subblocking is even more important because partial subblocking requires simpler OS support than superpages, reduces more effectively the page table sizes and is generally speaking more effective. The only limitation of partial subblocking is that the number of valid bits in a PTE does not allow for subblocking factors larger than 16. Techniques for supporting partial subblocking are essentially the same as those for supporting superpages.

**Complete subblocking** is even easier to support because a complete-subblock TLB entry has the same structure as a clustered PTE. In both cases, the system will have to deal with (a) block misses when a whole TLB entry is missing and (b) subblock misses when one of the page mappings in a TLB entry is missing. Subblock misses can be eliminated by prefetching all the page mappings of every new TLB entry brought into the TLB.

The same structure can also be used to support superpages and partial subblocking by storing in each entry of each bucket flags identifying the entry as a base page, a superpage or a partial subblock. Superpages that do not fit into a bucket could be handled by the replication method described above.