Guarded Page Tables for Alpha

Sebastian Schönberg

Dresden University of Technology
Faculty of Computer Science
D-01062 Dresden, Germany
email: schoenbg@os.inf.tu-dresden.de

Abstract

Modern processors don't use a hard wired mechanism to translate virtual addresses into physical addresses. Instead, software has to fill the translation buffer (TB). To achieve high performance, it is very important to find a fast algorithm.

Guarded Page Tables (GPT) are known to provide a fast translation from virtual addresses into physical addresses in large, sparsy filled address spaces. This paper describes an implementation of GPTs on an Alpha 21064 microprocessor including the adaption into PALcode, a very interesting feature of all Alpha machines to implement various strategies for hardware dependent requirements.

This paper describes an implementation of Guarded Page Tables as introduced in [3] and [4] on an Alpha. A similar implementation on MIPS machines can be found at [5].

1 Guarded Page Tables

In sparsy filled address spaces, many tables of an n-level page table tree are filled with only one entry and used to go on to the next level. Guarded Page Tables skip these intermediate levels using a guard. Figure **??** illustrates this. This method minimizes the time to translate a virtual address and saves memory.

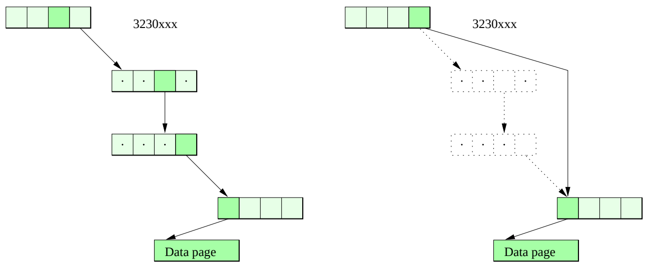

Figure 1: Guarded Page Table Tree

Furthermore, Guarded Page Tables support pages and page tables with different size of *size* = 2^n .

2 Translation Buffer

The Alpha 21064[1] contains two independent TBs, one for references to instruction pages (ITB) and one for data (DTB) pages. The ITB is twelve entries in size. Eight entries are used for the small 8KB pages; four entries are used for large 4MB pages. The ITB entry to be replaced is choosen by the processor for each region independently, using a Not Last Used (NLU) algorithm.

The DTB has 32-entries, which are used for all sizes of referenced data pages. In the same way as the ITB, DTB fill uses an NLU replacement algorithm. In further processor implementations, the strategy can be changed to a Robin Round (RR) algorithm.

All entries are untagged, but each one contains an Address Space Match (ASM) bit to exclude it from a TB flush operation. This avoids TB misses for pages which are shared between different address spaces after a context switch.

3 Translation Buffer Fill

Software, running in PALmode has to fill the TB in three steps. Depending on the cause of the error, one of the following entry points is invoked by the processor:

Name	Cause
nDTB-Miss	DTB Miss during access from native code
pDTB-Miss	DTB Miss while in PALmode
ITB-Miss	ITB Miss (can only occur in native mode)

Figure 2: 21064 PALcode TB Miss Entry Points

The steps are:

1. Write the original virtual address to the TB_TAG register. This writes the TAG into a temporary register and not the actual TB entry.

2. Write the PTE to the TB_CTL register to select the page size. Wait at least one cycle.

[1]Complete desciption see:[1] or [2]

3. Write the PTE to the TB_PTE register. This write causes both the TAG and the PTE fields to be written to the TB entry.

4 Page Table Structure

Each PTE is represented by two, 64 bit wide entries. The first one contains the guard, the required protection and the number of guard bits, which have to match the address for a valid translation. The second entry points either to the next level in the page table tree and codes the size of this table, or if the entry is a leaf entry in the tree, it contains the word that is written into the TB.

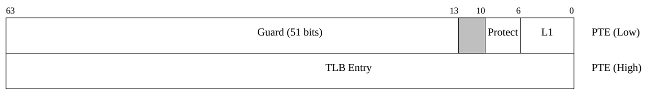

Figure 3: Guarded Page Table Entry

4.1 The Guard Field

The guard is a mask, which must match partly the virtual address at each stage in the page translation. Since the smallest page size supported by the Alpha is 8 KB, only the upper 51 bits are used for the guard.
Protection information is necessary to figure out whether a page access is valid or not. It must be included in each PTE to benefit from the advantages of hierachical page tables. This allows the changing of the protection of a range of n continuous pages by changing only one bit.
To adapt to all variants of information supported by the Alpha hardware the following bits are required:

- **User-Kernel:** decides which mode is required to access this page.
- **Executable:** this page contains instructions which can be executed.
- **Writable:** decides whether the page can be written or not.

The protection field is enhanced by one bit, called 'InTree' which decides between a leaf and a node of the tree. Bits 6..9 are used to store this information.
The lowest 6 bits are used to indicate how many bits of this guard are valid. The number is coded in the form of $64 - x$. Strictly speaking, it represents the number of bits remaining for the next page table tree level.

4.2 The Pointer Field

The pointer field contains two pieces of information: the size of the page table of the next level and the starting address of this table.
To indicate the number of entries of the next page table tree level, six bits are required. The information is stored in the form of $64 - \log_2 size$.
It is possible to use the lowest six bits without restricting the functionality. It can be assumed that all page table entries are aligned to their size (16 bytes), hence the lower 4 bits are always zero. If it is decided that page tables with less than 4 entries are useless and not required, then two further bits are zero for each base address of a page table.
The remaining 58 bits are the pointer to the next page table.
If this entry is a leaf of the tree, all 64 bits of the field contain the entry, which is written in the TB.

4.3 The Page Table Base Register

Translation must take place in PALcode, which provides 32 additional registers, called PAL_TEMP, which can be used for data storing only. One of these PAL_TEMP registers is dedicated to point to the root of the page tree. It is structured in the same way as the pointer to a PTE.

5 The Operation

The operation itself is very simple. Figure 4 shows the steps of the operation.

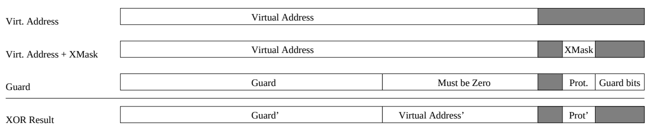

Figure 4: Guarded Page Table Operation

1. The protection bits of the virtual address are merged with the constant XMask. This sets bits 4..9 of the virtual address to a defined state.
2. The result is combined with the guard field by an XOR instruction. All bits which are identical in the guard and in the address will be cleared. If the guard' field is equal to zero, the address matches the guard.
3. By setting the XMask in the original virtual address, the resulting protection field prot' is not zero, if either the translation is finished (only the InTree bit is set) or the access is not permitted.

The algorithm can be written down in assembler code A.

6 Required Cycles

The code above was examined by the DEC PALcode Violation Checker (PVC) tool and cycles are measured with the Alpha internal processor Cycle Counter (CC).

The loop contains 12 instructions and takes 11 cycles + 2 stall. Two instructions can be issued in parallel. The complete code for an ITB miss, including parsing a three level page table, TB fill, entering and leaving PALcode takes 26 cycles for saving and restoring used registers, 3*13 cycles for the parsing loop and 18 for entering and leaving PALcode. On an 133 MHz 21064 Alpha, each TLB fill takes $0.6\mu s$.
The resulting calculated and measured 83 cycles compares favourably with the 400 cycles for other software implementations, like Mach.
All instuctions in the code are ordered very carefully, to minimize required stalls. This leads to a 25% faster version than the original one.
The measured case does not consider cache misses during load of page table entries.

7 Summary

The main advantages of Guared Page Tables are the less amount of required main memory to map few frames in a large address spaces.
As shown in the paper above a software implementation of Gurded Page Tables is able to fill TLBs with acceptable results of required time and instruction cycles. Features of PALcode, like additional storage (PAL_TEMP registers), non interruptibility, and code using physical addresses, allows a fast implementation on this processor.
Further work is required for translation prefetching, handling of second level TLBs, and support for multimedia applications, in particular mapping of continuous data streams.

References

[1] *DEC* DECchip 21064 and DECchip 21064A, Hardware reference manual 1994 *Digital Equipment Corporation*

[2] *Richard L. Sites* Alpha Architecture Reference Manual 1992 *Digital Equipment Corporation*

[3] *Jochen Liedtke* Some Theorems About Guarded Page Tables 1993 *GMD SET-RS*

[4] *Jochen Liedtke* Page Table Structures For Fine-Grain Virtual Memory 1994 *GMD SET-RS*

[5] *Jochen Liedtke, Kevin Elphinstone* Guarded Page Tables on MIPS R4600 1995 *UNSW-CSE-TR-9503*

A Assembler Code

```
HDW_VECTOR (PAL_ITB_MISS_ENTRY)
pal_itb_miss_entry:
        mtpr    a0, pt7                          pt7 is PAL_TEMP reg 7
        mtpr    t0, pt0
        mtpr    t1, pt1
        mtpr    t5, pt5
        mtpr    t6, pt6
        mfpr    a0, excAddr                      Get fault address
        mtpr    t3, pt3
        ldiq    t6, 0x3c0
        ldiq    t5, 0x1c0
        bis     a0, a0, t3

        mfpr    t0, ptPageBase                   t0 = PageBase
        bic     a0, t6, a0                       0x3c0 - clear prot. area
        mtpr    t2, pt2
        bis     a0, t5, a0                       XOR inverts only X, W,
        mtpr    t4, pt4
        bic     t0, 0x3f, t2                     t2 = old.frame start bas
        srl     a0, t0, t1                       t1 = table_bits (a0)
        ldiq    t6, PP_EXECUTABLE | PP_INTREE    required protection
1:
        s4addq  t1, 0, t1                        t1 = t1 * 4
        s4addq  t1, t2, t2                       t1 = t1 * 4 + t2
        /*stall*/
        ldq_p   t1, 0(t2)                        load lower part of gpte
        ldq_p   t0, 8(t2)                        t0 = higher part of gua
        /*stall*/
        xor     a0, t1, a0                       guard bits and flip prot
        bic     t0, 0x3f, t2                     t2 = old.frame start bas
        srl     a0, t1, t4                       shifted guard
        and     a0, t6, t1                       P - check
        bis     a0, t5, a0                       XOR inverts only X, W,
        bne     t1, itb_protection_miss          P - check
        srl     a0, t0, t1                       t1 = table_bits (a0)
        beq     t4, 1b
        br      zero, itb_guard_fault

itb_protection_miss:
        Depending on t1 bits error handling
        bic     t1, 0x40, t1                     PP_INTREE is always s
        bne     t1, itb_protection_fault

itb_translation_valid:
        bne     t4, itb_guard_fault
        mtpr    t3, tbTag                        Virt. Address
        mtpr    t0, tbCtl
        mfpr    t1, pt1                          Required cycle
        mtpr    t0, itbPte                       Page Table Entry
        mfpr    t2, pt2
        mfpr    t3, pt3
        mfpr    t4, pt4
        mfpr    t5, pt5
        mfpr    t6, pt6
        mfpr    a0, pt7
        mfpr    t0, pt0                          Restore Scratch Registe
        hw_rei
```