Inverted Page Tables

& more

1993 by Jerry Huck (HP), Jim Hays (ESS)

presented by VLM on 21. February 2005

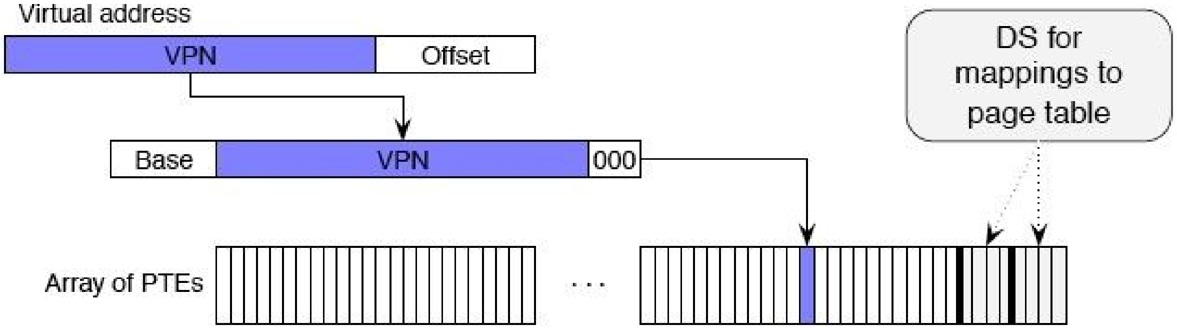
Overview

- 1. Review
  - 1.1 What is VM / Page Table?
    - 1.1.1 Linear PT
    - 1.1.2 Forward-Mapped PT
  - 1.2 What is a TLB?
- 2. Inverted (hashed) Page Tables
- 3. New TLB Techniques
  - 3.1 Superpage
  - 3.2 Subblocking
- 4. Clustered Page Tables
- 5. Evaluation
  - different PT designs, different VM designs
- 6. Conclusion

What is a Virtual Memory System?

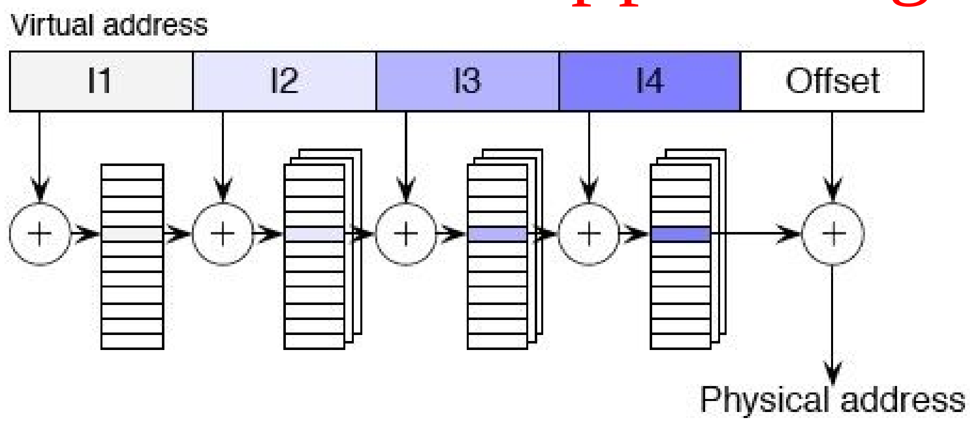
- 2nd virtual Address space, mostly larger and flat (32bit)
- One / Proccess
- VM System defines mapping VA → PA
- Includes mapping and Security Informtion
- Not single byte, but pages (4k or 8k), size of PT?
- Size too big to have 1 / Proc, two solutions
  - Linear PT
  - Forward-mapped PT

Linear Page Tables



- Single, huge array residing in VA space, bottom-up access
  - used by VAX-11 & MIPS R4000
- PT itself must be mapped seperately
  - Reside in PA or mapped via reserved TLB entries
  - Could use multilevel-trees or hashed PT to map themselves
- Multilevel Linear PT requires each intermediate node to be a page
- Needs 6 levels on 64bit systems
- OSF/1 on MIPS R3000 used 3 levels

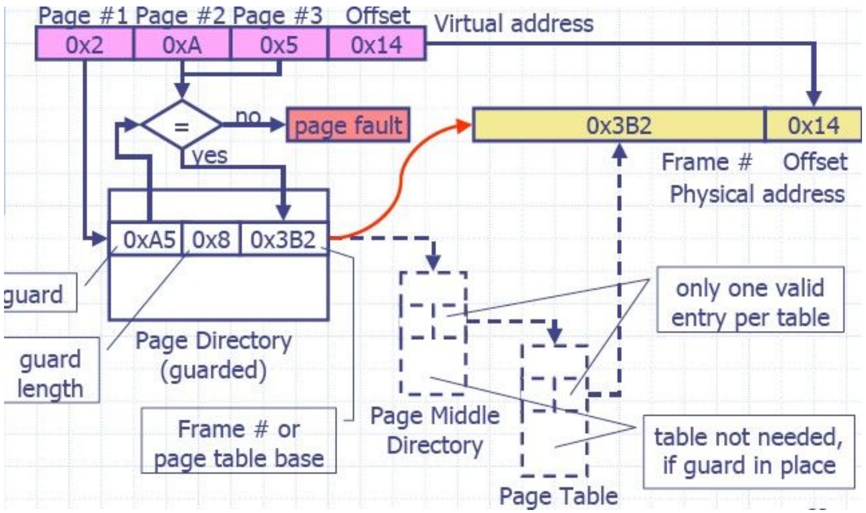
Forward-mapped Page Tables



- *n*-ary trees, top-down access, resides in PA space
- Each level uses fixed bit fields in VPN
- Intermediate nodes do not need to be a page
  - Can have different branching factors at different levels
- Performs bad on sparse address spaces & needs about 7 levels on 64bit, 2 solutions to shortcircuit:
  - Guarded PTs
  - Region Lookaside Buffers

Guarded PTs

- Collapses entire levels of tree
- Entry contains prefix & length
- Length field Gives OS great flexibility, can use different sizes of PTs & Ps
- Should balance tree



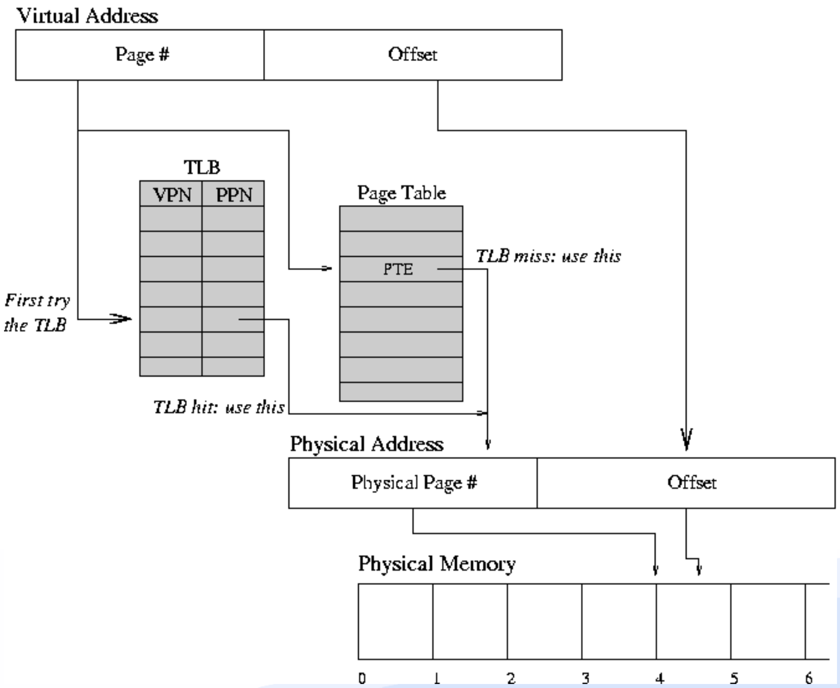
- can use *n*-assoc. for faster translation & further reducing size
- can also use RLB for Adr. in same Block to directly access deeper levels

Region Lookaside Buffers

- uses Buffer to store mappings to deeper PTs (Block locality)
- similar to reserved TLB entries for Linear PT
- used in HaL or called PTP cache in SuperSPARC

Translation Lookaside Buffer

- Fast, small buffer for complete V to P translations
- VPN is compared to tag, if match data gives the PPN
- includes valid bit
- normally set-associative to reduce conflict misses
  - makes it a little slower



HW TLB uses FSM
- inflexible

- SW TLB can support every PT struc., seperate kernel/user handling code
  - but inflict precise interrupt overhead, flushing of pipeline, reorder buffers, I-Cache etc.

Inverted Page Tables

- For 64bit Page Table ludicrously big, size?
- Needs ≥ 7 levels to keep Table size small
- Idea: PM size magnitudes smaller

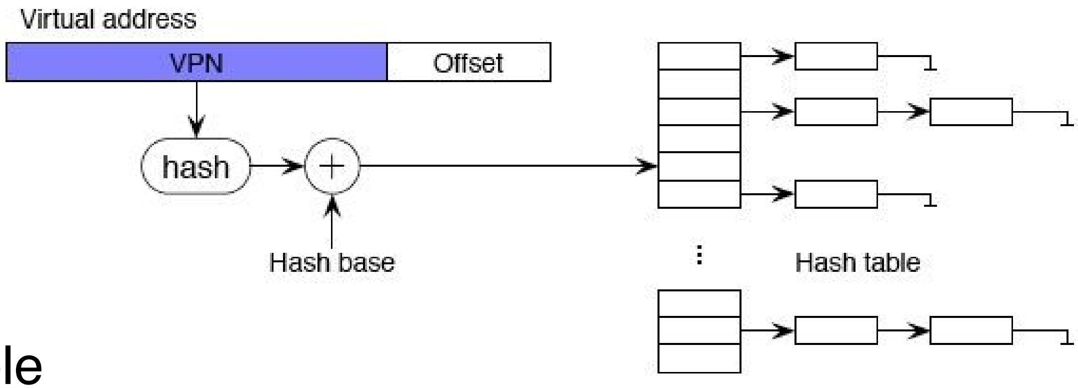
→ build table only for existing (physical) pages, index with PA

→ Entry gives VA

- 3 Problems:
  - IO devices that map into PA space create holes and waste space in table
  - can be fixed by only including mapped pages in table
    - → needs to search whole table for PA on TLB miss
  - No aliasing possible → use global aliases
- Solution
  - use Hash-Function on PA → Hashed Page Table

Hashed Page Tables

- Hash Funct. maps VPN to
  - Hash Anchor Table giving possible mappings (IBM System/38) or
  - directly to a bucket



- use chaining or Overflow Table
- fixed size only big enough to cover available Memory
- theory suggests prime-number size, while practice dictates power of 2
- includes PID in entry since Table is global
- for aliasing just add more than one entry in chain (PID)
- fixed, high (200%) overhead → good for sparse address spaces
- reduce next pointer by including only offset or avoid by multiple PTEs / Bucket (PowerPC)
- cut bits from VA – can be inferred since entries map to same bucket

New TLB Techniques - Superpages

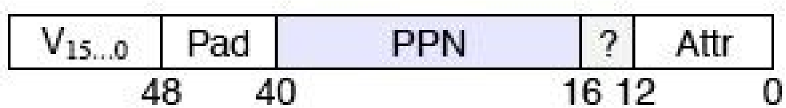
- Two approaches to reduce TLB miss ration and store mapping more compactly
- **Superpages**: Pages with power-of-two size of base page size
- Need to be aligned in both VA & PA space

Superpage mapping for size of 2^s

- Could be used for kernel/shared pages
- like LPTs/FMPTs good for dense, localized address spaces

New TLB Techniques - Subblocking

- **Complete-SB**: Several base pages managed with one TLB tag
  - stores multiple PPNs / tag ► increased data size
- includes all PPN, need to be only aligned in VA
  - introduces *Block* and *Subblock* misses



Partial-subblock mapping (subblock factor 16)

Partial-SB: stores only one PPN but multiple valid bits
- Need to be aligned in both VA & PA space
- but not all pages need to be mapped as in Superpages or CompleteSB

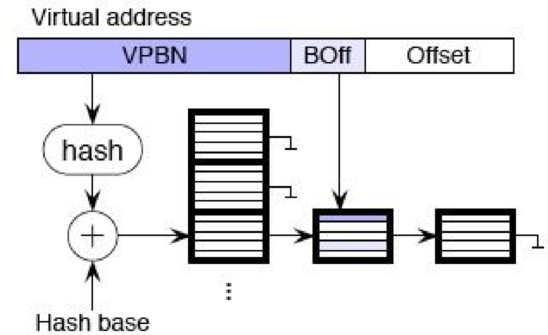
- How can we adapt these to Page Tables?

Adapting Superpages/Subblocking to PTs

- SP/SB useless if OS doesn't support them with proper mem. alloc and they are not replicated in the PTs, thus 3 Solutions:
- **Replicate PTEs**: store a superpage PTE @ every base page covered by the superpage
  - → space overhead, 16 PTEs for one 64k Superpage
- **Multiple PTs**: make one PT for every superpage size in use and search each for mapping
  - → smaller overhead, but takes longer
- **Linear/FM Nodes**: store Superpage pointer at intermediate nodes
  - → FMT can support any SP size by varying branching factor
  - → whereas LPT cannot

Clustered Page tables

- Similar to Hashed Page Tables
- Stores mappings for consecutive pages with a single tag (HPT with subblocking)
- Subblock factor can be chosen depending on address space sparsity
- Less overhead than HPT
- has fewer buckets / shorter list
  - ➔ improves access time
- can require more memory if mem. use is very sparse ► adjust subblock factor
- access time worse if PTEs span multiple cache lines
- first used by Solaris 2.5 on UltraSPARC

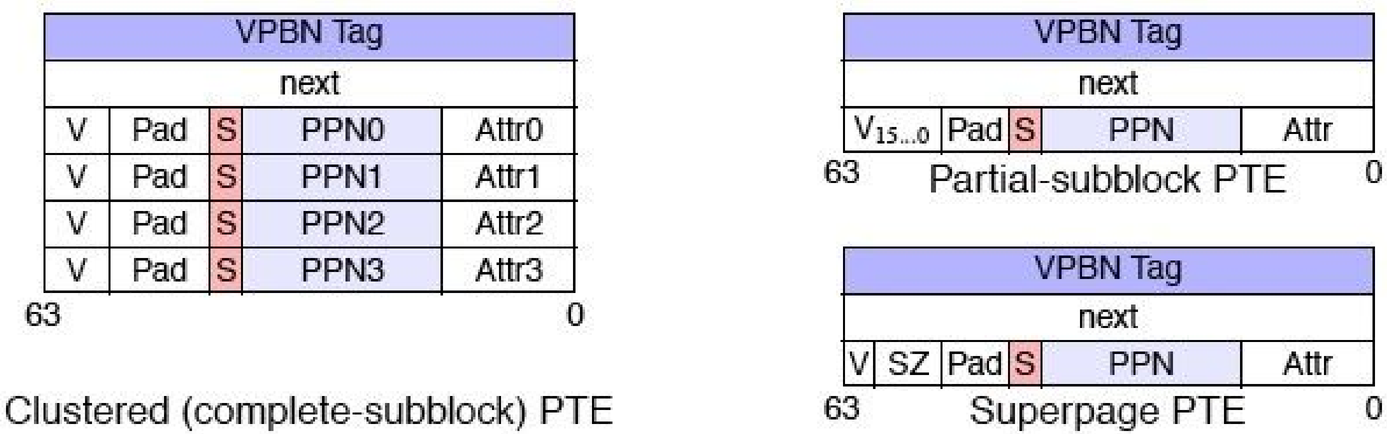


VPBN Tag			
next			
V	Pad	PPN0	Attr0
V	Pad	PPN1	Attr1
V	Pad	PPN2	Attr2
V	Pad	PPN3	Attr3

63 … 0

Clustered PTE (Subblock factor 4)

Partial-subblock and Superpage PTEs in CPTs



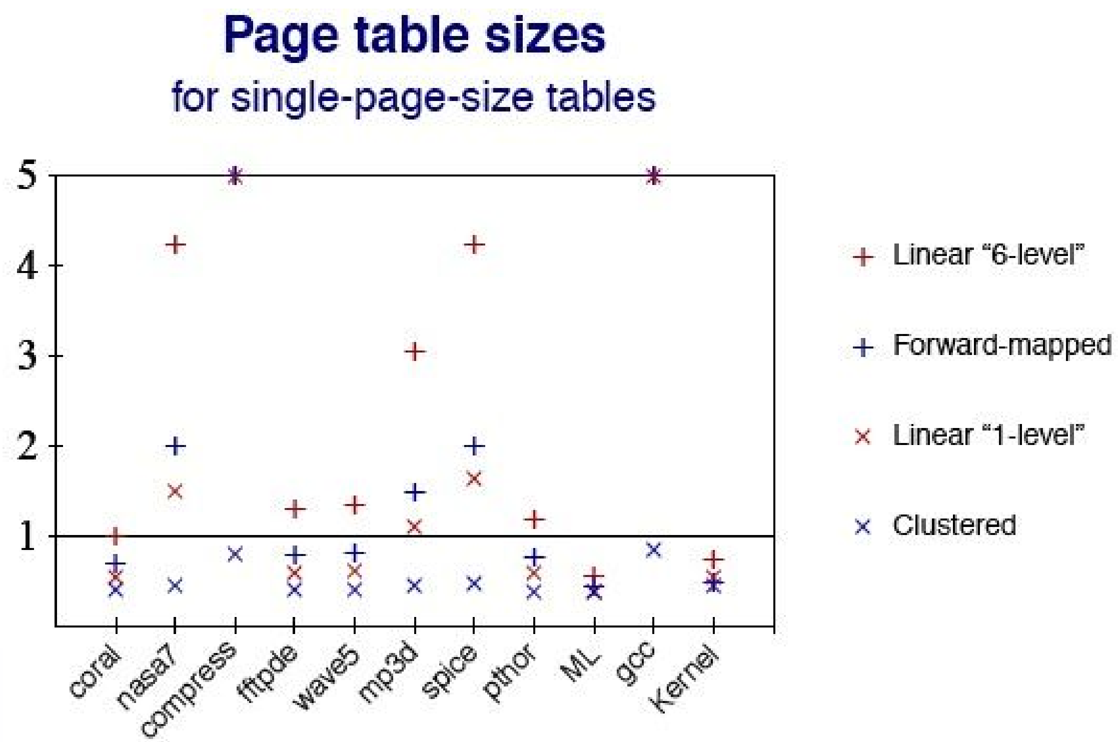
- CPT resembles complete-subblock TLB entry
  - CPTs can be enhanced to support partial-subblocking and superpages
- Use special flag (S field) to distinguish PTE types
- when partial-subblocking use same subblock factor as the CPT

Evaluation Setup 1

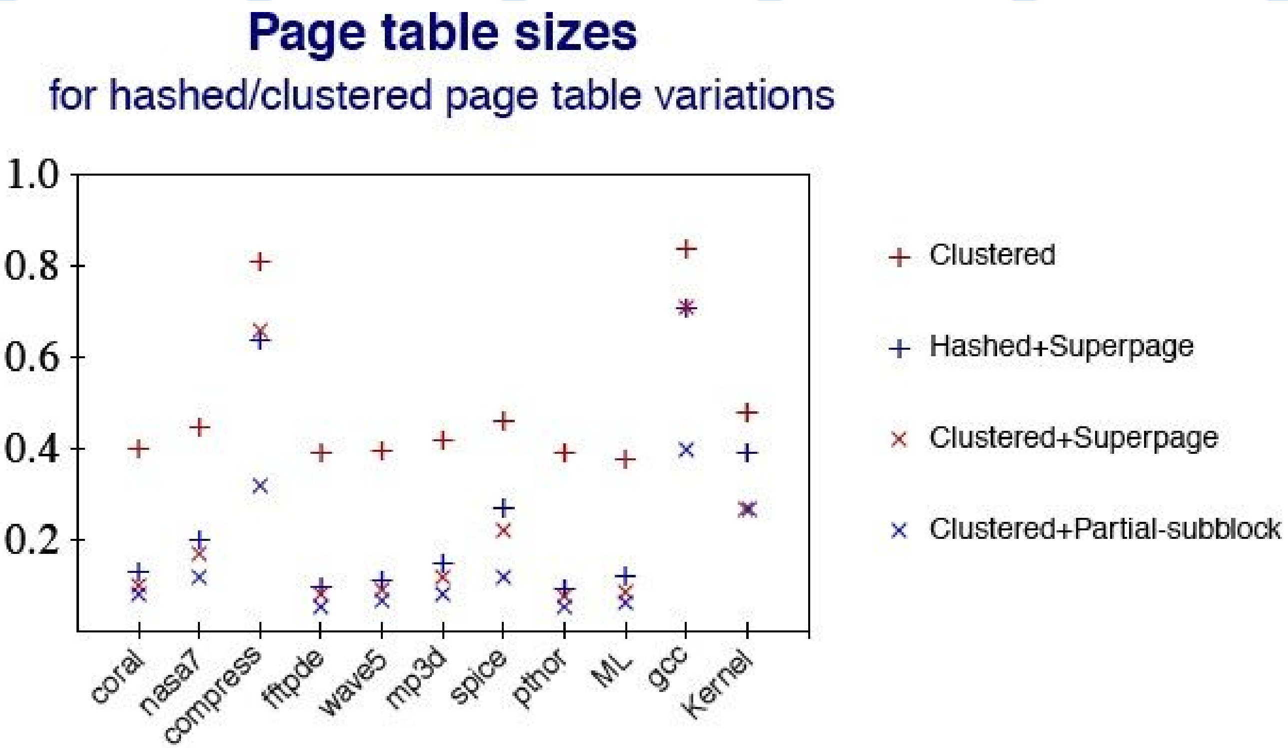
- Solaris 2.1 on a SparcServer 10, testing with 32bit workload
- Too complicated to implement all PT variations in real System
- Instead a TLB and PT simulator is built into kernel
- Studied aspects:
  - PT size
  - PT access time, by measuring the average number of chache lines accessed per TLB miss
  - Average number of cache lines per PT traversion

Page Table Size Eval

- Figure is normalized to HTP size
- "1-level" assumes that intermediate nodes take zero space
- Result: CPTs are smallest

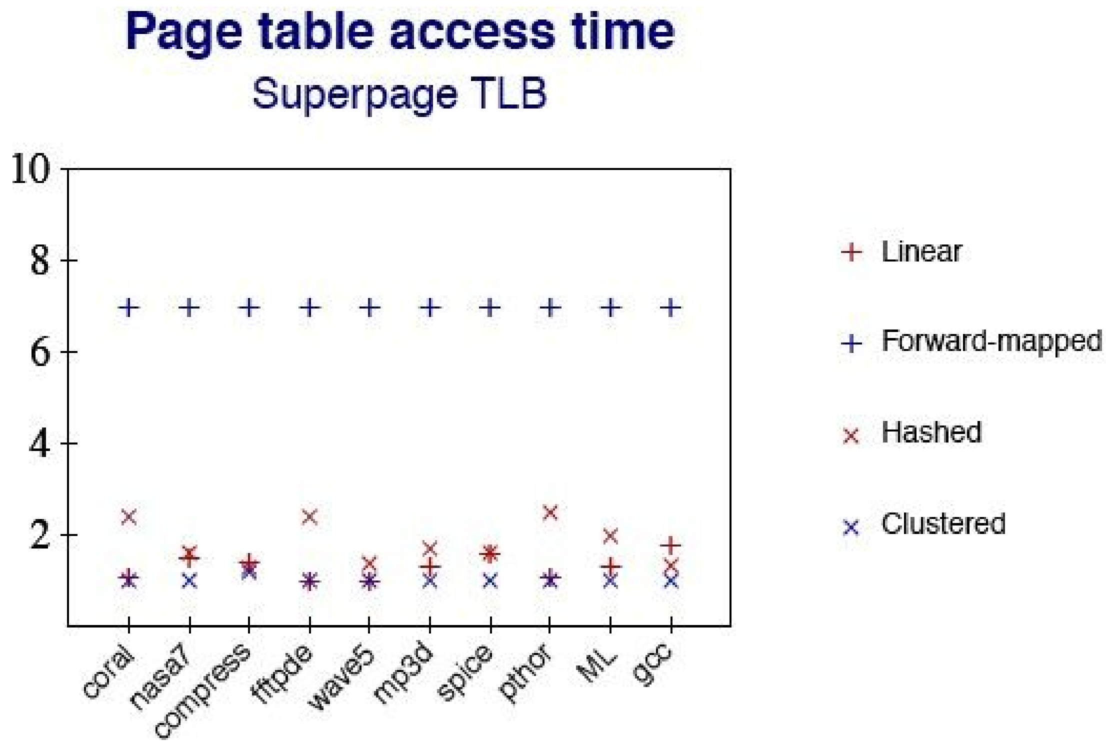


Page Table Size Eval 2



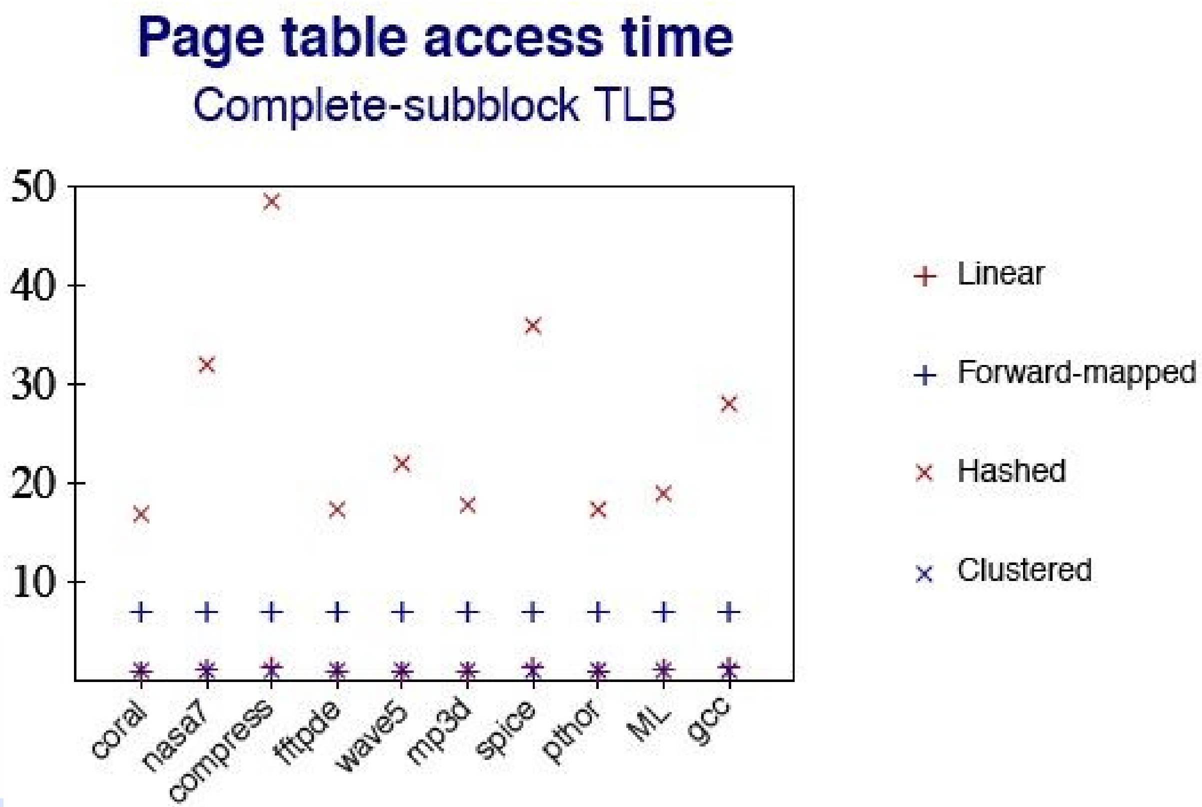
- using 4k base pages
- 64k superpages
- or subblock factor 16
- Result: CPT with partial-subblocking best

Page Table Access time Eval



- TLB fully associative
- 64 entries
- 4K base page size, 64k superpages
- again: CPTs best

Page Table Access time Eval 2



- Subblock factor 16; using complete-subblock prefetching
- HPT performs disatrous
- Complete subblock entries are sensitive to cache line size

Different VM System Organizations Eval

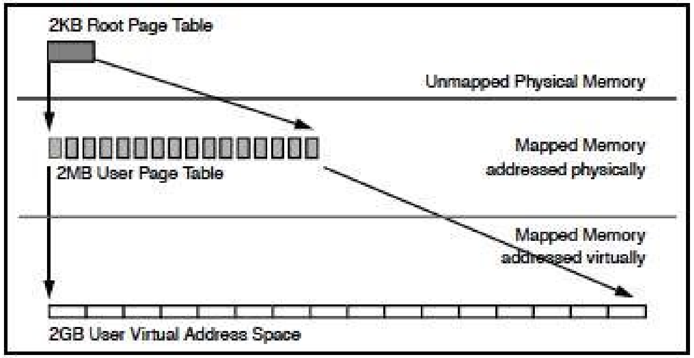


Figure 3: The BSD/Intel page table organization. The Intel page table is similar to the MIPS and NOTLB page tables; it is a two-tiered hierarchical table. However, unlike the other two, it is walked in a top-down fashion. Therefore, the user page table is a set of page-sized tables (4KB PTE pages) that are not necessarily contiguous in either physical space or virtual space (they do not need to be contiguous in virtual space because the table is never treated as a unit; it is never indexed by the VPN). These 4KB PTE pages map 4MB segments in the user's virtual address space. The 4MB segments that make up the user's address space are contiguous in virtual space.

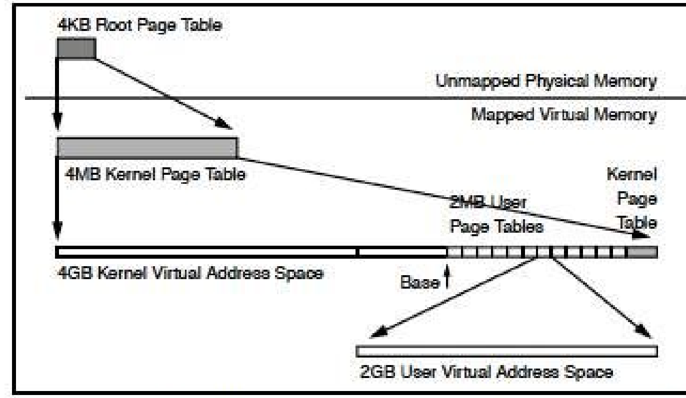


Figure 2: The Mach/MIPS page table organization. Mach as implemented on MIPS has a three-tiered page table. A user-level address space is mapped by a 2MB table in kernel space at an offset aligned on a 2MB boundary and related to the process ID of the user-level application: the virtual base address of the table is essentially *Base + (processID * 2MB)*. The top 4MB of the kernel's virtual address space is a page table that maps the 4GB kernel space. This kernel table is in turn mapped by a root table in physical memory.

- **Intel**: 2l, hirarchic Table, top-down
- HW, f-a. 128 I&D TLB
- 4kb not cont. PTs map 4mb in user space that is cont. in VAS
- 2kb root table maps these PDs

- **Mach/MIPS**: 3l Table
- split SW TLB
- User space mapped by aligned 2mb in 4gb kernel Table, which top 4mb map the whole table
- 4kb phys. root table maps top 4mb

Eval2 – MIPS + PA-RISC

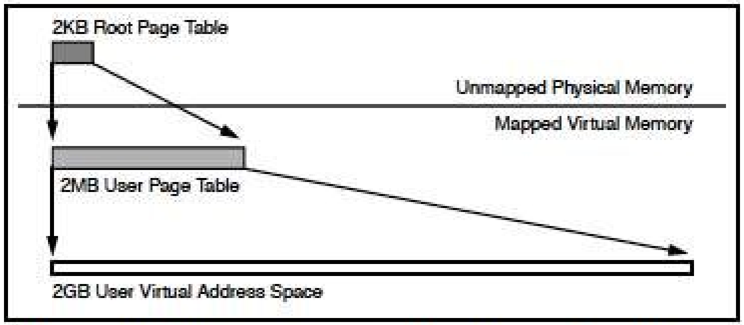


Figure 1: The Ultrix/MIPS page table organization. The Ultrix page table on MIPS is a simple two-tiered table. The user address space is the bottom 2GB of the hardware's address space; the top 2GB belongs to the kernel. A 2KB table wired down in physical memory maps each user page table.

- **Ultrix/MIPS**: 2l, bottum-up
- split SW TLB, 128 f-as. I&D TLB (16 res. for kernel mappings)
- 2gb user space mapped by 2mb LPT in VAS
- all User PDs mapped by 2kb root Table in PAS

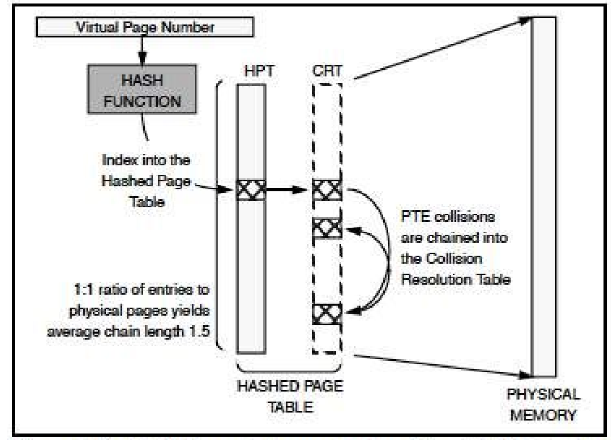


Figure 4: The PA-RISC page table organization. The PA-RISC hashed page table is similar in spirit to the classical inverted page table, but it dispenses with the hash anchor table, thereby eliminating one memory reference from the lookup algorithm. Since there is not necessarily a 1:1 correspondence between entries in the table and page frames in the system, the PFN must be stored in the page table entry, thereby increasing its size. While the collision-resolution table is optional, we include it in our simulation.

- **PA-RISC**: IPT with HAT, overflow table
- 128 f-a. I&D TLB
- 16b PTEs
- h-f: XOR of upper VA and lower VPN

Eval2 – Findings

- HW TLB (i.e. finite-state-machine) does not inflict so much overhead but is inflexible
- thus x86 organization is best, even with the 2 accesses/TLB miss
- when OS uses intelligent Page placing (dense VAS) IPTs can impact data chaches less than LPTs, although their PTEs are 4x bigger
- SW TLB miss handling can account for more than 10%, up to 40% of kernel execution time
- Taking everything into account, cache misses as result of VM moving data around, TLB miss handling, VM Interrupts etc. the total overhead of the VM System is about 10%-30%
- Intel and anti-technique (purely SW) NO-TLB least dependet on Int. when caches grow larger

→ Precise interrupt handling need more attention, because of VM

→ future VM organizations should use SW programmable HW TLBs and HPTs/CPTs because of 64bit

Conclusion

- Conventional PT mechanisms not practical for 64bit
- FMPTs almost worthless in 64bit even with shortcircuiting
- LPTs have low overhead and miss penalty
  - ► could work when mappings to table itself are hashed
- Subblocking & Superpaging increase TLB Hit ratio and most PTs can be changed to support both techniques
- LPTs / FMPTs are acceptable on dense address spaces
- HPTs better for sparse address spaces
- CPTs augment HPTs with Subblocking & Superpaging and are even more efficient

► CPTs best known solution for big (64bit) address spaces

Questions?