

Department of Computer Science Institute for System Architecture, Operating Systems Group

PAGE TABLE STRUCTURES FOR FINE-GRAIN VIRTUAL MEMORY

JOCHEN LIEDTKE

CARSTEN WEINHOLD



MOTIVATION

- Context: object oriented systems
- Objects are pieces of memory
- Access control, based on:
 - Segments
 Not really there, huge tables
 - Pages Hierarchies, sharing, coarse-grained
- Goal: purely page-based architecture that efficiently supports fine granularity



REQUIREMENTS

- Fine-grained address spaces need:
 - Small pages (e.g., 16 byte)
 - Various page sizes (powers of 2)
 - Different page sizes can be mixed freely
 - Efficient operations on hierarchy of small and large regions



RSITAT MULTI-LEVEL PTS

- Conventional page tables:
 - Multi-level
 - Few fixed page sizes (4K, 2M, 4M, ...)
 - Large tables per level (512, 1024 entries)
 - Coarse-grained
 - Wasted storage for sparse address spaces
 - "Small" address spaces are expensive



INVERTED PTS

- Inverted page tables:
 - One entry per page
 - Sparse address spaces are efficient
- Hashed page tables:
 - IPT entries with physical address \Rightarrow aliasing
- Problems:

TU Dresden

- No hierarchy, changing attributes expensive
- Fixed page sizes, high load on TLB

Page Table Structures For Fine-Grain Virtual Memory



GUARDED PTS

- Sparse address space:
 - Only one valid entry in 2nd and 3rd PT level
 - PTs effectively not needed, wasting memory

Idea:

TU Dresden

- Remove unneeded PTs
- Add extra info to PT entries to indicate shortcut

Page Table Structures For Fine-Grain Virtual Memory



Figure 4: Guarded Page Tables

6





- PT entries augmentated:
 - Bit string g: "guard"
 - Size of next PT / data page
 - Length of g, size of next level can vary
- Translation works like for conventional PTs
- Conventional PTs are special case: $g=\emptyset$





Figure 5: Conventional Translation Step.



Figure 6: Guarded Translation Step.

When using only standard ALU operations for decomposing the virtures, checking, the guard apprefix and synthesizing, the physical address slation step would require multiple cycles. It seems reasonable to spe

can always he differentiated, because a binary page table with one nil entry is a singunitary page table with one nil entry is guard. Therefore, the resulting tree has only the minimal number of binary nodes needed to reach all k pages. We start at the root and distribute the colles logs that here? Genifstenty is needed of the page y points to another binary page table, we add half of its cost to the costs of each Che two filtries: of address-space / page sizes



Therefore, 2k entries are enough for k reachable pages. This upper bound of the TV Dresden is independent of the address-space size and of the set of ⁹ reachable pages. We call this property *small* and try to hold all trees small



16 16 2 GUARDED PAGE TA The performance of the performance of the performance it is end The performance of the performance it is end to achieve high performance it is end lation steps. To achieve high performance it is essential to reduce the lation steps. To achieve high gent of all the performance it is essential to reduce the of the page-table tree, he page table tree.

- Translation steps
 It turns out that, of translate all high so that all pages have a depth of at most no end to be an in-pit of at most no end transform all nodes which decode 2 or more bits:
 It turns out that, of translate all high so that all pages have a depth of at most no end transform all nodes which decode 2 or more bits:
 It turns out that, of translate all high so that all pages have a depth of at most no end transform all nodes which decode 2 or more bits:
 It turns out that, of translate all high so that all pages have a depth of at most no end transform all nodes which decode 2 or more bits:
 It turns out that, of translate all high so that all pages have a depth of at most no end transform all nodes which decode 2 or more bits:
 It turns out that, of translate all high so that all pages have a depth of at most no end transform all nodes which decode 2 or more bits:
 It turns out that, of transform these binary nodes with their direct successors into quate the page of a stransform these binary nodes with their direct successors into quate nodes which decode 2 or more bits:
 It and dross bits can be
 It and dross bits can be all the dross bi
- *n* address bits can be translated in $\lceil n/2 \rceil$ steps
- All nodes that decode only one bit can be transformed





The steps can further be reduced to n/4 by n The steps can2f2rBner be associative/ **Translation** parallel trans-

Transformed tree remains No other cases require transformations. Since Small
No other cases require transformations here size, the resulting tree is also small since formations increases the tree size, the resulting tree is also small since formations increases the tree size inscretive Transformations. Since

Page Table Structure 3 or kinassociative a Translation

TU Dresden





- Translation in n/2 steps still takes long
- *k*-associative translation: ^{Figure 7:} 4-associative Guarded Translation

 $\blacksquare \text{Perform } k \text{ translation} \\ k = 4 \text{ and } 8 \text{-byte entries, a } 32 \text{-byte path is required (as in a low of the low of the$



A detailed description of k-associative translation

TECHNISCHE UNIVERSISATE Entry-Point Caching ZATIONS

Fortunately, we can drastically reduce the average number of steps by caching side entry points into the translation tree. We divide the address space into equal-sized regions and cache the optimal entry points of accessed regions (see figure 8). Side Entry-Point Caching



Figure 8: *Side Entry Translation*. Page Table Structures For Fine-Grain Virtual Memory

TU Dresden



USER MAPPINGS

- Paper proposes user-level mapping
- PT entries augmented with type *t*:
 - t = alias
 - Address in PT entry is virtual
 - Aliasing of virtual memory
 - *t* = call-on-reference
 - Address in PT entry is function pointer
 - Executes user-defined code on access



22





Figure 11: Physical Aliasing.







MAP INSTRUCTION

- Unprivileged map instruction:
 - Allows app to modify its own PT entries
 - No kernel entry / exit needed
 - Only works if PT entry type t matches
 - Virtual addresses only, no isolation breach



FINAL SLIDE

- GPTs allow for a wide range of page sizes
- Very efficient for sparse address spaces

- Discussion points:
 - What about PT allocation?
 - Different page sizes vs. complexity?
 - Usecases?
 - Practical performance? MIPS prototype?