CMP SCI 635: Modern Computer Architecture

Hash, Don't Cache (the Page Table)

Name: Kunjal Panchal Date: 24th Oct, 2019 Student ID: 32126469

Paper: 13 – Virtualization [Yaniv16]

Strengths:

1. Pointed out the flaws of Itanium Hash-based table design and then proposed the optimizations. Especially clustering, which can help with spatial locality, which was a major advantage of radix-based tables.

2. The performance of radix page tables depends on PWCs, so their effectiveness degrades when workloads access larger and larger memory regions with lower locality. In contrast, the optimized hashed page tables cut the page walk overhead without resorting to PWCs, as their performance is unaffected by the memory footprint or the access pattern of the application.

Weaknesses:

1. Hash tables are spread over a large memory area. This necessitates sharing the page table between all processes; killing a process requires a linear scan of the hash table to find and delete the associated PTEs. Marking deleted slots has the undesirable side effect of a longer hash table lookup, because it no longer depends solely on the load factor.

2. A paper on "Variable Radix Page Table: A page Table for Modern Architecture" by National ICT Australia presents a new page table structure, the "variable radix page table", which overcomes many of the disadvantages of other page table structures. Unlike a hashed page table, the variable radix page table naturally accommodates shared segments and mixed page sizes.

   For this paper, the shared segments and mixed page sizes are accommodated by adding another layer, as implemented in the Power architecture. The Power architecture requires a two-level translation procedure for each memory reference. But multiple page sizes may boost performance with fewer virtual to physical translations, at the cost of potential waste of memory that was allocated but never used.

   In this case, variable radix page table might be the better choice.

Questions/Assertions:

1. The MMU searches for an entry with the longest matching tag, to shorten the page walk as much as possible. But won't this increase the search time as first we look for the lowest level of PTE and if nothing is found, we look for second lowest level and so on?

2. Can memory protection be separated from address translation via a protection lookaside buffer or memory protection unit. E.g., IBM-360 "storage protection"? and does this work for virtual memory?

3. The downside of a physically addressed cache is that the MMU has to sit between the processor and the cache, so the cache lookup is slow. L1 caches are almost never physically addressed. But if the operating system takes care that different processes use non-overlapping address spaces, can we do away with virtual addressing?

4. There is some ambiguity about look-"ahead" and look-"aside" buffer. Few forums claim that look-"ahead" is the analysis in advance of subsequent decisions that would be made if a particular branch of an algorithm was followed. But doesn't TLB also take care of that kind of spatial locality? Is there really any difference between the two terms?

5. With Virtual Addressing, if the cache is not fully flushed between context switch (other process's data can exist in cache), that is an aliasing problem. Same memory can be directed from the different virtual address. But sometimes aliasing is expected, e.g., where running privileged kernel code mapped in the same virtual address range of all processes. Would hash tables have any scope of aliasing?