Contents

1 Literature Review

1.1 Huge Pages

Increasing TLB reach can be achieved by using larger page sizes, such as huge pages[1], which are common in modern computer systems. The x86-64 architecture supports huge pages of 2 MB and 1 GB, backed by OS mechanisms like Transparent Huge Pages (THP)[2] and HugeTLBFS in Linux. However, available page sizes in x86-64 are limited, leading to internal fragmentation issues. For instance, allocating 1 MB with 4 KB base pages requires 256 PTEs, but using a 2 MB huge page would waste half of the memory space. Some architectures offer more page size choices, such as Intel Itanium, which allows different areas of the address space to have their own page sizes. Itanium uses a hash page table to organize huge pages, but without significant changes to the conventional page table, it only helps reduce page walk overheads. HP Tunable Base Page Size permits the OS to adjust the base page size, but still faces internal fragmentation problems, with HP recommending a base page size of no more than 16 KB. Shadow Superpage[3] introduces a new translation level in the memory controller to merge non-contiguous physical pages into a huge page in a shadow memory space, extending TLB coverage. However, this approach requires all memory traffic to be translated again in the memory controller, resulting in additional latency for memory accesses.

1.2 Direct Segment

Early processors often used segments to manage virtual memory, where a segment[4] essentially mapped contiguous virtual memory to contiguous physical memory. Unlike pages, which are relatively small, segments can be much larger, offering the potential for more efficient memory management in certain scenarios. This concept of segmentation has seen a resurgence

in some modern approaches that aim to enhance translation coverage by designating specific areas in the virtual address space.

This method allows programmers to explicitly define a single segment for applications requiring significant memory. It introduces two new registers to the system, which indicate the start and end of this segment. Virtual addresses within this segment are translated by calculating the offset from the virtual start address and applying this offset to the physical start address. This straightforward method simplifies the translation process for large memory areas but requires significant modifications to the source code of applications.

1.3 Range Memory Mapping (RMM)

Redundant Memory Mappings (RMM)[5] enhance memory management by introducing an additional range table that pre-allocates contiguous physical pages for large memory allocations, creating ranges that are both virtually and physically contiguous. This approach simplifies address translation within these ranges by adding an offset, similar to Direct Segment, but RMM supports multiple ranges and operates transparently to programmers, requiring no source code modifications. The range table, separate from the conventional page table, holds the mappings for these large allocations. To determine which range an address belongs to, RMM compares the address against all range boundaries, a process that is computationally expensive and therefore performed only after an L1 TLB miss. To optimize this, RMM uses a range TLB (RTLB) to quickly identify if an address falls within any pre-allocated range, facilitating efficient translation and reducing overhead. Range mapping works alongside the paging system by generating TLB entries on TLB misses and still performing TLB lookups for each virtual address translation. Unlike traditional segmentation mechanisms, range mapping activates a range lookaside buffer (RTLB) located with the last level TLB upon a miss. The hardware TLB miss handler then searches the RTLB for the miss address and, if found, generates a new TLB entry with the physical address derived from the base virtual address and range offset, along with permission bits. If the RTLB also misses, the system defaults to a standard page walk while a range table walker simultaneously loads the range into the RTLB in the background, avoiding delays in memory operations. The RTLB, functioning as a fully associative search structure, ensures that most last level TLB misses are handled efficiently by range mapping, reducing the need for costly page table walks.

1.4 CHERI

CHERI (Capability Hardware Enhanced RISC Instructions) extends conventional processor Instruction-Set Architectures (ISAs) with architectural capabilities to enable fine-grained memory protection and highly scalable software compartmentalization. CHERI is a hybrid capability architecture that can combine capabilities with conventional MMU (Memory Management Unit) based systems. The contributions of CHERI include:

- ISA changes to introduce architectural capabilities.
- New microarchitecture proving that capabilities can be implemented efficiently in hardware, with support for efficient tagged memory to protect capabilities and compress capabilities to reduce memory overhead.
- A newly designed software construction model that uses capabilities to provide fine-grained memory protection and scalable software compartmentalization.
- Language and compiler extensions for using capabilities with C and C++.
- OS extensions to support fine-grained memory protection (spatial, referential, and (non-stack) temporal memory safety) and abstraction extensions for scalable software compartmentalization.

References

[1] A. Panwar, S. Bansal, and K. Gopinath, "HawkEye: Efficient fine-grained OS support for huge pages," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, pp. 347–360. [Online]. Available: https://dl.acm.org/doi/10.1145/3297858.3304064

[2] J. Navarro, "Practical, transparent operating system support for superpages."

[3] C. H. Park and D. Park, "Aggressive superpage support with the shadow memory and the partial-subblock tlb," *Microprocessors and Microsystems*, vol. 25, no. 7, pp. 329–342, 2001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141933101001259

[4] A. Basu, J. Gandhi, J. Chang, M. D. Hill, and M. M. Swift, "Efficient virtual memory for big memory servers," *SIGARCH Comput. Archit. News*, vol. 41, no. 3, p. 237248, Jun. 2013. [Online]. Available: https://doi.org/10.1145/2508148.2485943

[5] V. Karakostas, J. Gandhi, F. Ayar, A. Cristal, M. D. Hill, K. S. McKinley, M. Nemirovsky, M. M. Swift, and O. Ünsal, "Redundant memory mappings for fast access to large memories," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. ACM, pp. 66–78. [Online]. Available: https://dl.acm.org/doi/10.1145/2749469.2749471