A revision on different techniques for efficient address translation and Superpages

Mariana R. Hernández Rocha
Master in Computer Science
ITAM
Mexico City, Mexico
mhern224@itam.mx

***Abstract*—This survey analyzes different strategies to increase the *translation lookaside buffer* (TLB) coverage. Some of the reviewed work have solutions built on previous ones; others propose different approaches and attempt to cover specific tasks. However, they all have in common the presence of Superpages in their system.**

***Index Terms*—virtual memory, pages, superpages, TLB.**

I. Introduction

Most of the architectures that use virtual memory pages require a *Translation Lookaside Buffer (TLB)* to accelerate the process of address translation. The TLB is a cache memory that contains the most recently used translations among physical and virtual memory addresses.

Historically, physical memory has increased its size following an exponential growth and, combined with increasingly complex applications, require more memory usage; hence, the current TLB coverage may be insufficient.

The rest of this article is organized as follows: Section II presents a general review of different techniques to manage Superpages to increase TLB. Section III revises the evolution of the technology that improves the TLB performance and describes the different approaches. In Section IV, an analysis of the different proposals is presented. Section V presents a brief conclusion and wonders what is next for Superpages.

II. Increasing TLB coverage

The naive solution to increase TLB coverage would be to increase the TLB size. However, this is not an attractive one since processors' actual requirements are fast TLB accesses, even supporting multiple cache accesses per cycle. So, enlarging the TLB size would lead to losing its "cache" characteristic. Here is where *superpages* come in. A superpage is a memory page made of power-of-two multiples of base pages; they occupy one line of the TLB, the same as base pages. Moreover, several architectures support them.

This next section in this work is a chronological revision of tools that increase TLB coverage. Many of the articles here presented demonstrate that, with the correct adequations, Superpages are an excellent solution to:

1) Increase the amount of memory that the TLB can access.
2) Reduce misses in the TLB.

III. Superpage approaches

Architectures that support superpage usage also require operating system support. The OS support used for virtual memory with a single fixed page size comprises a virtual memory manager, file systems, a physical memory manager, a page replacement process, and a hardware-dependent layer. However, it is possible to modify this support's functionality to obtain a more efficient performance of superpaging. The articles that will be reviewed within this section present their modifications to the existing support.

A. Trade-offs in supporting two-page sizes

This pioneer approach [1] explores how to increase the TLB reach by increasing the Superpage size and allowing the system to choose from two available Superpage sizes. They examined these alternatives experimentally and found that using medium-size Superpages (defined as 32KB Superpages) reduces the CPI_{TLB} but increases the working set size 60%. The usage of two Superpage sizes (4KB and 32KB) combined only increases the working set size by about 10%. A fully associative TLB is also compared against a two-way set-associative TLB, finding that the fully associative results were comparable to the large page size. In contrast, the set-associative results were, at best, comparable to using 32KB pages.

B. Surpassing the TLB performance of Superpages with less operating system support

In [2], *Talluri et al.* explain the benefits and costs of working with medium-size Superpages, defined as Superpages in the order of 64KB. A system that modifies and improves the existing support for Superpages was developed. To create this system, they introduced a page-size assignment policy and page promotion/demotion mechanisms. The *page-size assignment policy* is in charge of deciding the page size for each virtual address and whether the policy should choose statically or dynamically. The *page promotion* mechanism consists of creating a larger superpage upon coalesced base pages. In this operation, the system needs to allocate physically contiguous memory to copy the base pages. As expected, this *gather* operations has a high cost. The *page demotion* mechanism works in a similar way to page promotion, but it creates a smaller superpage. However, adding policies to

this support system has several consequences. First, it adds overhead. Secondly, this system is very invasive as data structures (page table), and interfaces must be redesigned. So, [2] introduce an adaptation of an existing concept in cache memory, *sub-blocking*, distinguishing between *complete* and *partial* subblock TLB entries. The complete subblock TLB entries are more extensive than a superpage TLB entry, increasing access time. This disadvantage introduces partial subblock TLB entries, which solve this problem by coalescing the four pairs of physical page numbers (PPN) and attribute fields in a single pair.

Despite the benefits of *partial subblock TLB entries*, this only works if the operating system offers support for aligned physical memory allocation. So, a new *page reservation* algorithm is proposed to allocate memory to help TLB. The *page reservation* algorithm uses a *free list* and a *reserved list*.

C. Reducing TLB and memory overhead using online superpage promotion

In [3], several policies are proposed. After analyzing past misses, these policies can determine when to promote a superpage. During runtime, this analysis considers two counters:

- prefetch: this counter will increment on a miss if the potential Superpage contains the current fault page and one or more TLB entries.
- capacity: these charges are related to the TLB miss stream. The capacity counter will increment if the potential Superpage coalesces TLB entries and, this prevents the currently referenced page from being previously evicted from the TLB.

The policies presented by *Romer et al.* are the following:

- **OFFLINE**: Selects Superpages to promote greedily; the Superpages with the highest ratio of TLB misses cycles eliminated to the promotion cost in cycles will be promoted.
- **ONLINE**: this algorithm decides when to promote a superpage based on the counter's prefetch and capacity.
- **APPROX-ONLINE**: this algorithm works the same as ONLINE, but it maintains prefetch charges only, so it does not generate the overhead of maintaining the capacity counters.
- **ASAP**: from the acronym of *as-soon-as-possible*, this algorithm promotes as soon as all the base pages have been referenced no matter how frequently they were referenced.

D. Increasing TLB reach using Superpages backed by shadow memory

The mechanism proposed by *Swanson et al.*[4] creates Superpages from discontiguous base physical pages. When addressable physical memory exceeds the installed physical memory, the *Memory Controller TLB* (MTLB) uses a portion of the unused physical address range to virtualize the physical memory a second time. This method allows us to create Superpages of arbitrary size from discontiguous fixed-size pages; this is done by what they call *shadow* pages, which are regions of physical address space for which the MTLB re-maps accesses to real physical pages.

When there is an attempt to access a virtual address, if the access misses a cache fill request with the shadow address, it will be sent to the *Memory Management Controller* (MMC). Then, the MTLB determines if the address is real or a shadow.

This mechanism addresses the fact that the mapping of a superpage must be on appropriately sized, aligned, and contiguous physical memory, determining which regions are suitable to use Superpages. It reduces the need for moving full Superpages from/to disk.

E. Reevaluating online superpage promotion with hardware support

Since the analysis made by Romer *et al.*, two new architectural trends emerged: superscalar and many other memory system designs. *Fang et al.* [5] combined *Swanson's* Impulse system with a modified *Romer's* simulation system, and they use it to evaluate two of Romer's policies: ASAP and APROX-ONLINE.

It found that competitive promotion policies have better results only for a copying mechanism. For a mechanism based on re-mapping, it is more suitable to use policies that promote Superpages as soon as all the base pages have been touched/referenced.

F. Practical, transparent operating system support for superpages

In [6], *Navarro et al.* uses the *reservation-based* superpage of *Talluri et all.*.

Navarro et al. support multiple superpage sizes; the system will use the buddy allocator to find a suitable range of contiguous base pages. A policy makes the selection of the preferred Superpage size; the policy will tend to choose the maximum Superpage size to avoid relocating in case of choosing a too small Superpage.

Another feature of this approach is that they incorporate preemption. To control the fragmentation of memory, the buddy allocator coalesces available regions of memory. Finally, they modified the page replacement daemon to be contiguity-aware. Promotion and demotion are done at an increasing pace, with speculative demotion.

To keep all these tasks working, Navarro *et al.* use two different data structures: lists and population maps. The lists allow the system to have a record of reserved page frame regions that are not fully populated. The population map is a more complex tree-like structure that maps the virtual address to a page frame that was probably already reserved for that address.

G. Concurrent Support of Multiple Page Sizes on a Skewed Associative TLB

Since TLB is a cache memory, the authors of this approach[7] adapt the concept of *skewed associative caches* to the functionality of the TLB. Typically, an n-way set-associative cache has n distinct bank. A line of data with

address X will be physically mapped to physical line $f(X)$ in any bank. On the other hand, the *skewed associative cache*[8] has different mapping functions for the cache bank, for the line address X, the physical line where it will be mapped is $f_0(X)$ in cache bank 0 or, $f_1(X)$ in cache bank 1.

The skewed associative TLB has three properties:

1) Each virtual address has a possible and unique mapping location in each way.
2) Every way has its mapping function.
3) For any supported page size, applications using some page size can use the whole**(overall)** space of the TLB.

H. Hardware support for superpage coalescing

The next solution is patented hardware support for Superpage usage [9]. Having many cache memory levels in the memory hierarchy leads to a model of main memory called non-uniform memory array (NUMA), which distributes the main memory among many processor clusters. This hierarchy means that the systems will contain multiple memory controllers, limiting, even more, approaches like *Fang et al.*. This work's motivation was to find an improved method for Superpage coalescing that could be adapted to NUMA systems.

The system allocates a range of contiguous physical memory pages. The memory controller then moves the virtual memory pages to the new page mapping using a state engine. Finally, the memory controller permits access to the virtual memory pages as they get copied to the physical pages.

The memory controller has a mapping table that temporarily contains the old and new page entries, and after copying them, it releases them so they can be accessed.

I. Supporting superpage allocation without additional hardware support

This document[10] aboard the fragmentation problem proposing a placement policy. The main idea of this approach is that the OS should not depend on contiguous allocations; it should be capable of using small pages and, only at convenience, use Superpages.

The placement policy is called *grouping pages by mobility* (GPBM). The policy's objective is to reduce external fragmentation when necessary but allow Superpages to allocate without affecting the allocator by grouping pages of the same mobility type within a matching type arena.

Four mobility types (movable pages, reclaimable pages, temporary pages, and non-reclaimable) are defined; the address space is split into arenas of the largest Superpage size. Since the whole system performance depends on the allocation method, this policy is based on a *binary buddy allocator*.

The authors also propose *age-biased arena reclaim*, a modification to the page reclamation algorithm that is contiguity-aware.

J. Colt: Coalesced large-reach tlbs

In [11], the authors exploit contiguity in tens to hundreds of pages are contiguous. The operating system already produces this contiguity level using extra structures, intruding the operating system, or requiring prescribed amounts of contiguity as previous Superpage systems.

Coalesced Large-Reach TLBs (CoLT) have three different variants; each is a hardware mechanism that makes the TLB coalesce multiple contiguous virtual to physical address translations, and the three of them follow the next principles:

1) Detect instances of consecutive address translations.
2) The mechanism will coalesce only on TLB misses.
3) Coalescing must be unintrusive.

The three mechanisms presented in *Pham et al.* work are CoLT-Set Associative (CoLT SA), CoLT Fully Associative (CoLT FA) and, CoLT All.

K. Scattered superpage: A case for bridging the gap between superpage and page coloring

In [12], *Chen et al.* state that to achieve a better performance in multi-core systems, they must consider the LLC and TLB. For the LLC optimization, *Chen et al.* suggest using *page coloring*. *Page coloring* is a software approach used to partition shared cache. In this method, physical memory is divided into colors; each color maps to a different cache set, partitioning the cache memory.

Page coloring technique only works if there are overlapping bits between *cache set index* and *page number*. Meanwhile, when using Superpages, the *Superpage offset* would cover all the bits in the *cache set indes*, and it would cause the OS not to control the page color bits. Every Superpage would occupy all the available page colors.

A virtual Superpage will be scattered mapped into multiple physical Superpages, using just the region specified by color in the physical Superpage. Hence, multiple of those color regions constitutes a physical Superpage.

L. Increasing TLB reach by exploiting clustering in page translations

In [11] the CoLT system was introduced, finding that an OS typically show contiguous spatial locality in a scale smaller than Superpages. In [13], the authors study the spatial locality among the translations from virtual to physical memory. Later, they propose a multi-granular TLB organization that exploits that clustered spatial locality. For the analysis, the characterization of spatial locality in page tables was done by observing that translations were also nearby in the same address region. Eleven benchmark races were analyzed.

They relaxed the clustered spatial locality definition, considering that virtual-to-physical page table entries coalesce as long as they are nearby to introduce their multi-granular TLB model. The multi-granular TLB has a *clustered TLB* for page table entries with clustered spatial locality, a *conventional TLB* for page table entries with no clustered spatial locality, the *coalescing logic* detects clustered spatial locality and its inner *logic* performs look-ups, evictions, etc.

M. Supporting Superpages in non-contiguous physical memory

In [14], *Du et al.* propose a *gap-tolerant sequential mapping* (GTSM), an approach to Superpage construction from non-contiguous physical memory. The problem inspires Their solution that page retirement represents to obtain contiguous physical memory when creating Superpages. They utilize a different format for the page table to map the Superpages.

What GTSM does, is divide a virtual Superpage into multiple smaller memory blocks, each of a fixed size, these blocks are bigger than a base page, and a set of them form a memory slice. Then, the system maps the blocks to memory using a block selection bitmap.

They also present *gap-tolerant page directory entry* (GTPDE), an extension of GTSM whose results reduces page table size by 50%. Although this seems like a straightforward and practical approach, it is very restricted by allowing just one Superpage size.

N. Prediction-based superpage-friendly TLB designs

After analyzing the TLB behavior of various workloads, the authors of this work[15] were able to propose a lightweight binary Superpage mechanism that attempts to predict ahead of time if memory access is to a Superpage or not. This mechanism leads to the article's goal: to enable the design of an *elastic* TLB_{pred} design that dynamically adapts its page capacity to fit the application's need.

TLB_{pred} is a set-associative cache that uses two different indices: one for 8KB-based and another Superpage-based. All Superpages share the same indexing bits regardless of their size; thus, they are free to use all sets.

The authors also consider various page size prediction alternatives and evaluate the *skewed TLB* design of *Seznec et al.* Even when the prediction-based TLB design showed a better performance with more hits and efficiency, this article lacks experimental results for the proposed TLB_{pred}.

O. Large pages and lightweight memory management in virtualized environments: Can you have it both ways?

The coarse granularity of Superpages limits lightweight memory management systems so, it has been observed that the virtualization software disjoin Superpages into base pages, losing the advantages that Superpages represent in terms of address translation overhead. However, the splinted Superpages often maintain the original continuity and alignment in virtual and physical address spaces.

In [16], *Pham et al.* propose a hardware solution called *generalized large-page utilization enhancements* (GLUE), it identifies the memory regions of the base pages that were disjoined from a superpage.

Their work characterizes the conflict between Superpages and lightweight memory management in different hypervisors, architectures, and containers. They also propose an interpolation-based TLB speculation to improve performance and, finally, study GLUE trade-offs.

P. Coordinated and efficient huge page management with Ingens

For this work[17], the authors identified that processors had few TLB entries reserved for Superpages, so they were inspired by the need to modernize memory management.

Ingens is a memory manager for the operating system and hypervisor that does not interfere with workloads that already have good performance with the current Superpage support. This manager is based on two principles:

- Memory contiguity is a resource that must be allocated across processes.
- Information regarding spatial or temporal accesses is the raw material to manage contiguity.

They monitored space and time with the mechanisms *Util bitvector* and *Access bitvector*.

The authors also modified the page fault handler to promote Superpages after a superpage region accumulates enough allocated base pages. Superpages can also be demoted, but high utilization Superpages are not demoted because it deteriorates performance. To maintain the physical memory fragmentation-free, Ingens monitor the physical memory state and compacts the memory to reduce latency. It also has information about access frequency to balance page sharing with performance and monitors and distributes memory contiguity between processes, so the distribution is fair.

Q. Efficient Address Translation for Architectures with Multiple Page Sizes

In this article[18], the authors are proposing a TLB design that covers three requirements:

1) The hardware should be almost completely utilized and avoid misses.
2) The design should be energy efficient.
3) TLB look-ups, miss handling, and fill should not be time-consuming, so the implementation must be simple.

To achieve number 2, the TLB must be set-associative. Using this approach is problematic since TLBs need the page size on look-ups, but identifying the virtual page number requires the page size.

The proposed MIX TLB uses a single set-indexing scheme for translations of all page sizes. The problem here is that they use bits within the superpage page offset to choose the TLB set, so a superpage is mapped to multiple TLB sets. They call this *mirroring*. To solve this, they took advantage of the contiguous allocation of virtual and physical addresses that the OS does. They detect and coalesce the into the same TLB entry and, if they coalesce enough as the number of mirror copies, they balance the redundancy.

The implementation of MIX TLB yields a performance improvement from 10% to 30%; it responds to the rigidity of existing in TLB capacity allocation.

R. Making huge pages actually useful

The authors of this article present Illuminator[19] as a response to previous approaches to handling memory fragmentation. After analyzing memory management systems, they

found that incorrect handling of kernel pages is the root cause of common superpage-related problems like high CPU utilization and latency outliers. They identified two types of problems:

- Memory contiguity is polluted with unmovable pages, fragmenting the memory.
- The kernel migrates pages from polluted regions, inducing latency from the cost of recovering from fragmentation.

Illuminator tracks all the fixed pages to help the subsystems to avoid unnecessary work using those pages. The solution to the previously stated issues is approached by diving memory into three sets: unmovable region, movable region, and hybrid page block region. This solution is similar to the work of *Navarro et al.* with the creation of *pools*. Both approaches showed promising results. In the case of this article, they were improving the application performance up to 2.3 times.

S. SEESAW: Using Superpages to improve VIPT caches

The work of *Parasar et al.*[20] is inspired by *virtually-indexed and physically tagged* (VIPT) caches. These structures surged as a solution for L1 cache memories that require access to the TLB before cache look-up. However, VIPT caches require the index bits to be included in the page offset field; this constrains the sets that could be implemented in the cache memory.

VIPT caches were designed when systems accepted only one-page size, so the authors explored the opportunity of using Superpages to create VIPT caches with good performance, energy-efficient, and simple to implement.

The alternative proposed is called *Set Enhanced Superpage Aware* (SEESAW) caching; their solution supports three types of look-ups:

1) CPU look-ups for data in superpage.
2) CPU look-ups for data in a base page.
3) Coherence look-ups.

SEESAW showed a 3-10% improvement in run time and 10-20% improvement in memory access energy. However, the results change due to memory fragmentation, as mentioned before, a common problem in superpage usage.

T. Supporting Superpages and Lightweight Page Migration in Hybrid Memory Systems

In [21], *Wang et al.* focuses on *dynamic random access memory* (DRAM) and *non-volatile memory* (NVM) memory systems, which rely on page migrations to improve their performance and energy efficiency. It is necessary a lightweight page migration scheme to move pages that are frequently accessed from the NVM to the DRAM.

The authors study how to exploit superpage usage to increment TLB coverage while supporting lightweight page migration in hybrid memory systems. It is crucial to identify lightweight most frequently accessed pages and evaluate the impact of lightweight page migration on TLB coverage.

They propose *Rainbow*, a memory management mechanism that links Superpages and lightweight page migration for DRAM/NVM memory systems. Rainbow manages NVM and DRAM using different page granularities; it supports different page sizes using split TLBs. It can move base pages from a superpage to the DRAM without affecting the whole superpage TLB's integrity.

As this approach is taking advantage of hybrid memory systems' advantages, it is somewhat difficult to compare with other approaches. However, it reduces TLB misses by 99.9% and the address translation overhead, also improving application performance.

IV. Analysis

Talluri et al. in their first work, their findings depended on the OS mechanism to choose from the available Superpage sizes, architectural specifications, and a lack of knowledge on multiprogramming behavior. However, that article opened the door for new problems regarding operating systems. In the second work that they presented, they confirm their theories about Superpages.

Navarro et al. took many of the previous ideas and combined them, adding extra support to unify a system that efficiently uses Superpages for increasing TLB coverage. They obtained a good performance, though this solution is pretty complicated, and data structures were introduced overhead.

The first approach presented by *Seznec et al.*, is constrained by the experiments' architecture. It could be used to implement medium size partially associative L1 TLB or partially associative large size L2 TLB.

The mechanisms created by *Pham et al.* detect the intermediate level of contiguity that OS memory allocation mechanisms already get and demonstrated that the contiguity obtained by their mechanisms reduces 40% to 50% of TLB misses. However, this level of continuity is not enough to generate Superpages. Their second contribution studies the clustered spatial locality that already exists in applications. The performance eliminated 46% of L2 TLB misses and had a 7% CPU cycle reduction at best.

The approaches mentioned above were the most influential of all, though each of the articles revised in Section III made contributions to this topic.

V. Conclusions and Future work on Superpages

Using Superpages for increasing TLB coverage has been demonstrated to be the right solution for the bottleneck problem in virtual-to-physical translation problems. However, it requires much adaptation to exploit its virtues. The real challenge to Superpages has been to find a way to adapt it to different architectures and systems. We do not know what will come next in architectural trends that, perhaps, will require Superpages to modify its behavior or bring different solutions for address translation.

References

[1] M. Talluri, S. Kong, M. D. Hill, and D. A. Patterson, "Tradeoffs in supporting two page sizes," in *Proceedings*

of the 19th annual international symposium on Computer architecture, 1992, pp. 415–424.

[2] M. Talluri and M. D. Hill, "Surpassing the tlb performance of superpages with less operating system support," *ACM SIGPLAN Notices*, vol. 29, no. 11, pp. 171–182, 1994.

[3] T. H. Romer, W. H. Ohlrich, A. R. Karlin, and B. N. Bershad, "Reducing tlb and memory overhead using online superpage promotion," in *Proceedings of the 22nd annual international symposium on Computer architecture*, 1995, pp. 176–187.

[4] M. Swanson, L. Stoller, and J. Carter, "Increasing tlb reach using superpages backed by shadow memory," *ACM SIGARCH Computer Architecture News*, vol. 26, no. 3, pp. 204–213, 1998.

[5] Z. Fang, L. Zhang, J. B. Carter, W. C. Hsieh, and S. A. McKee, "Reevaluating online superpage promotion with hardware support," in *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*. IEEE, 2001, pp. 63–72.

[6] J. Navarro, S. Iyer, P. Druschel, and A. Cox, "Practical, transparent operating system support for superpages," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 89–104, 2002.

[7] A. Seznec, "Concurrent support of multiple page sizes on a skewed associative tlb," *IEEE Transactions on Computers*, vol. 53, no. 7, pp. 924–927, 2004.

[8] ——, "A case for two-way skewed-associative caches," *ACM SIGARCH computer architecture news*, vol. 21, no. 2, pp. 169–178, 1993.

[9] E. Elnozahy, J. Peterson, R. Rajamony, and H. Shafi, "Hardware support for superpage coalescing," Mar. 22 2007, uS Patent App. 11/551,168.

[10] M. Gorman and P. Healy, "Supporting superpage allocation without additional hardware support," in *Proceedings of the 7th International Symposium on Memory Management*, ser. ISMM '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 41–50. [Online]. Available: https://doi.org/10.1145/1375634.1375641

[11] B. Pham, V. Vaidyanathan, A. Jaleel, and A. Bhattacharjee, "Colt: Coalesced large-reach tlbs," in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2012, pp. 258–269.

[12] L. Chen, Y. Wang, Z. Cui, Y. Huang, Y. Bao, and M. Chen, "Scattered superpage: A case for bridging the gap between superpage and page coloring," in *2013 IEEE 31st International Conference on Computer Design (ICCD)*. IEEE, 2013, pp. 177–184.

[13] B. Pham, A. Bhattacharjee, Y. Eckert, and G. H. Loh, "Increasing tlb reach by exploiting clustering in page translations," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, 2014, pp. 558–567.

[14] Y. Du, M. Zhou, B. R. Childers, D. Mossé, and R. Melhem, "Supporting superpages in non-contiguous physical memory," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2015, pp. 223–234.

[15] M.-M. Papadopoulou, X. Tong, A. Seznec, and A. Moshovos, "Prediction-based superpage-friendly tlb designs," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2015, pp. 210–222.

[16] B. Pham, J. Veselỳ, G. H. Loh, and A. Bhattacharjee, "Large pages and lightweight memory management in virtualized environments: Can you have it both ways?" in *Proceedings of the 48th International Symposium on Microarchitecture*, 2015, pp. 1–12.

[17] Y. Kwon, H. Yu, S. Peter, C. J. Rossbach, and E. Witchel, "Coordinated and efficient huge page management with ingens," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 705–721.

[18] G. Cox and A. Bhattacharjee, "Efficient address translation for architectures with multiple page sizes," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 435–448. [Online]. Available: https://doi.org/10.1145/3037697.3037704

[19] A. Panwar, A. Prasad, and K. Gopinath, "Making huge pages actually useful," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018, pp. 679–692.

[20] M. Parasar, A. Bhattacharjee, and T. Krishna, "Seesaw: Using superpages to improve vipt caches," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 193–206.

[21] X. Wang, H. Liu, X. Liao, J. Chen, H. Jin, Y. Zhang, L. Zheng, B. He, and S. Jiang, "Supporting superpages and lightweight page migration in hybrid memory systems," *ACM Trans. Archit. Code Optim.*, vol. 16, no. 2, Apr. 2019. [Online]. Available: https://doi.org/10.1145/3310133