A Swap-based Cache Set Index Scheme to Leverage both Superpage and Page Coloring Optimizations

Zehan Cui[†‡], Licheng Chen[†‡], Yungang Bao[†], Mingyu Chen[†]
[†]State Key Laboratory of Computer Architecture, Institute of Computing Technology, CAS
[‡]University of Chinese Academy of Sciences
Beijing, China, 100190
{cuizehan,chenlicheng,baoyg,cmy}@ict.ac.cn

ABSTRACT

We propose a novel cache set index scheme called SWAP (swap-based cache set index). SWAP introduces a pseudo-physical address space that is used by the operating system. The real physical address used for cache and main memory access is obtained by simply swapping some of superpage number bits with cache set index bits from the pseudo-physical address. By adding a level of indirection to the physical memory management, we simultaneously support both page coloring and superpage optimizations. These work together to improve TLB and shared LLC performance with negligible cost. Our results show that SWAP can improve performance by an average of 15.1% (by up to 25.2%) compared to 7.34% and 8.26% for superpage and page coloring, respectively.

1. INTRODUCTION

Both the translation lookaside buffer (TLB) and last level cache (LLC) are critical to system performance, particularly as the working set sizes of applications and the number of cores on chip increase. The TLB accelerates virtual-to-physical address translation by caching frequently used page table entries (PTEs). However, TLB misses are expensive, since page walks to look up PTEs in main memory are required before accessing cache. Recent work has shown that TLB misses can degrade performance by 5-14% for nominally-sized applications [3]. In the era of "big data", this number will degrade to 50% as application memory footprints increase [18]. The LLC is usually shared by all cores in today's multicore architectures. However, uncontrolled LLC sharing leads to an inter-core contention problem [5, 22, 28] — one core may constantly evict another core's useful data. This problem grows as core number increases.

Superpaging [27, 26, 9, 20, 2, 1] is a straightforward method to reduce TLB misses by using large page sizes that increase the TLB reach[1]. A superpage, which spans a large number of contiguous small pages, can be mapped into one TLB entry, which alleviates conflicts for TLB resources.

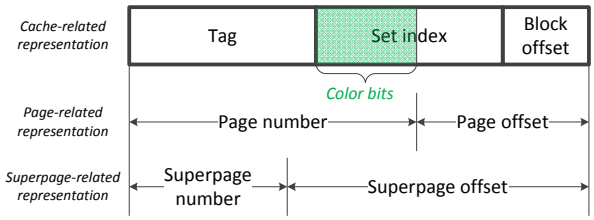

Figure 1: Different representations of a physical address. The overlapped bits of the cache set index and the page number can be controlled through OS virtual-to-physical memory mapping. There are no overlapped bits between the cache set index and the superpage number.

Page coloring [29, 13, 4, 24] is a software technique to map virtual memory to specific cache positions through selective physical page allocation. Figure 1 shows its principle: the page number field of the physical address is overlapped with the cache set index field. The OS can control the overlapped bits, or *color bits*, to map a page to specific cache sets. Page coloring based cache partitioning [28, 14, 30, 15, 17, 7] maps the memory of different applications to non-intersecting cache sets, which can eliminate inter-application cache contention.

Both superpaging and page coloring based cache partitioning are practical approaches that effectively alleviate TLB conflicts and LLC contention in real systems. However, there exists a contradiction when attempting to adopt the two techniques simultaneously, as shown in Figure 1: the superpage number field has no overlap with the cache set index, therefore no OS-controllable color bits exist to perform cache partitioning while using traditional superpaging.

In this paper, we propose a swap-based cache set index scheme, SWAP, to leverage both superpaging and page coloring based cache partitioning. SWAP adds another layer of memory space, called *pseudo-physical memory*, between virtual memory and physical memory. The OS manages this new memory space and maintains the mapping from virtual memory to pseudo-physical memory in the page table. A virtual address is first translated into a pseudo-physical address through the TLB. Then, a simple bit-swap operation is performed on the pseudo-physical address to generate the real physical address, which is then used to access cache and physical memory. The bit-swap ensures that some bits of the

[1]TLB reach is the amount of memory accessible from the TLB, equal to (TLB size)*(page size).

DAC'14, June 01–05 2014, San Francisco, CA, USA

http://dx.doi.org/10.1145/2593069.2593078.

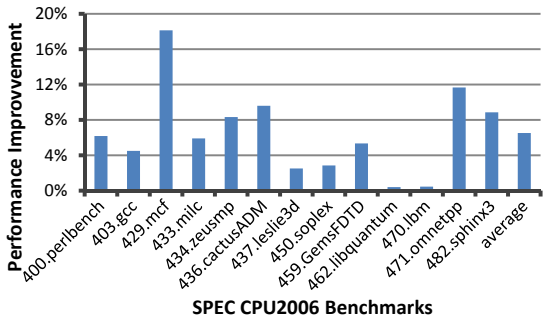

Figure 2: Performance improvement of memory intensive benchmarks when using 2MB superpage compared to 4KB page.

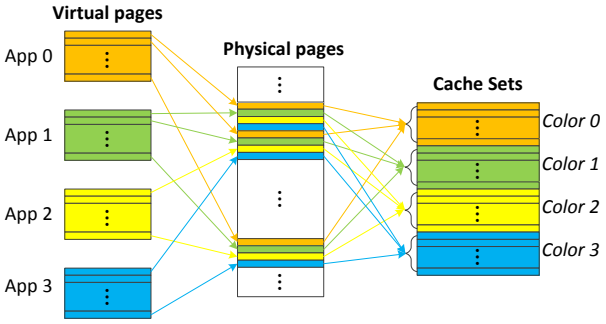

Figure 3: An example of cache partitioning to map different applications to non-intersecting cache sets (assuming four colors).

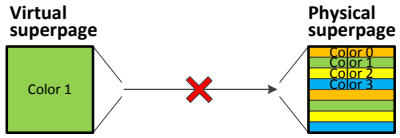

Figure 4: The incompatibility between superpaging and page coloring. A physical superpage occupies all colors (assuming four colors).

superpage page number are swapped with some bits of the cache set index so that the OS can now map superpages to specified cache sets by manipulating the superpage number in pseudo-physical space.

Our contributions are:

- We propose a simple swap-based scheme (SWAP) to bridge the gap between superpaging and page coloring based cache partitioning so that these two techniques can be simultaneously adopted to improve both TLB and LLC performance.
- We evaluate SWAP in a cycle-accurate full-system simulator, and show that it can supercede the benefits of either superpage (7.34%) or cache partition (8.26%), achieving on average 15.1% (and up to 25.2%) improvement in system performance.

2. BACKGROUND AND MOTIVATION

2.1 Superpaging

Superpages are supported in almost all processor architectures, including MIPS, Alpha, UltraSPARC, PowerPC, and x86. For example, the Intel SandyBridge microarchitecture has 64-entry instruction and data TLBs and a 512-entry secondary TLB per core for 4KB pages, but it also includes dedicated 32-entry TLBs to support superpages of sizes 2MB and 1GB [11]. By increasing page size, fewer page table entries are required for a given memory footprint, and conflicts for limited TLB resources are alleviated. Figure 2 shows the performance improvement when using 2MB superpages compared to 4KB page for memory-intensive SPEC CPU2006 benchmarks (under the simulation environment described in Section 4). We achieve an average speedup of 6.5%, which demonstrates the effectiveness of superpages. We observe the largest improvement for 429.mcf (18.1%), which has the largest footprint (1.6GB) and random memory access patterns. The TLB miss rate of 429.mcf can be reduced from 20.5% to 5.95% by utilizing 2MB superpages. Programs with high locality, e.g., 462.libquantum, tend to benefit little from superpages, since the TLB miss rate with 4KB pages is very small.

2.2 Page Coloring Based Cache Partitioning

Figure 1 shows that there are some common bits, the color bits, in the cache set index and page number (small page) fields. With k color bits, physical pages and cache sets can be divided into 2^k colors. A page can only be cached in sets with the same color. This lets the OS map a physical page with specified color to a process, a thread, or even an object. Figure 3 shows a typical example of partitioning the LLC for different applications to eliminate interference.

In most cases, only the shared LLC needs to be partitioned. The color bits should thus exclude the cache set index of private caches and only include overlapped bits between the LLC set index and page number. Otherwise, a core may only use a portion of its private cache. For the experimental platform described in Section 4, the set indices of the L2 cache and LLC are bits 6-14 and bits 6-18, respectively, so the color bits for the LLC partitioning are bits 15-18.

2.3 The Incompatibility between Superpaging and Page Coloring

Though both superpaging and page coloring are effective methods to address TLB conflicts and LLC contention, their implementations are traditionally incompatible. The intuitive reason for the gap is shown in Figure 1 — there are no common bits between the LLC set index and the superpage number. The insight is that a physical superpage, which spans a large contiguous memory space, usually occupies all colors — mapping to all cache sets — as shown in Figure 4. Thus, it is impossible to assign a specific color to a virtual superpage by directly mapping a physical superpage to it.

Chen et al. have proposed a *scattered superpage* scheme to bridge the gap between superpaging and page coloring [6]. Scattered superpage scheme adds calculation logic to the TLB to map one virtual superpage into parts of multiple physical superpages, and the OS can control the mapping from superpages to cache sets precisely. However, this will increase the latency of TLB lookups, which are on the critical path. Besides, scattered superpages require the OS to

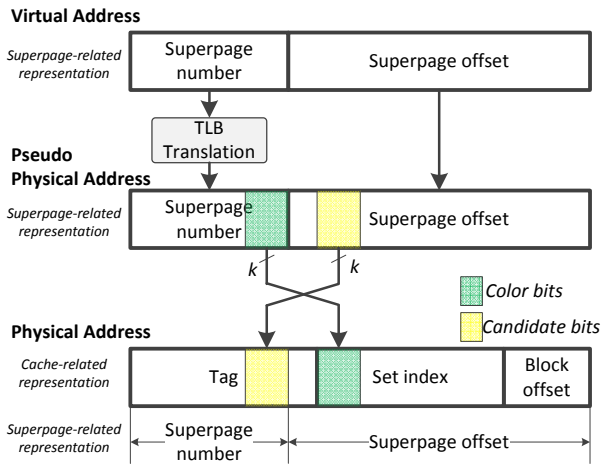

Figure 5: The swap-based scheme

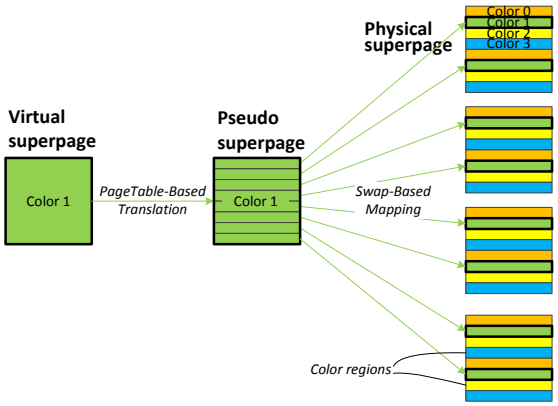

Figure 6: An example of mapping one pseudo-superpage into corresponding color regions in four physical superpages (assuming four colors).

maintain free color regions[2] of all physical superpages and to manage the mapping of color regions for each virtual superpage.

3. SWAP: SWAP-BASED CACHE SET INDEX

3.1 Pseudo-Physical Memory Space

As discussed above, the primary cause of the incompatibility between superpaging and page coloring is that a virtual superpage is directly mapped to a physical superpage that spans a large contiguous region of physical memory and thus occupies all colors.

Our proposal is to decouple the direct mapping relation between virtual and physical superpages by adding another layer of memory space — *pseudo-physical memory*. The OS can only see the pseudo-physical memory space. The OS manages the allocation and reclamation of pseudo-physical memory in the same way that it managed real physical memory space. The pseudo-physical memory can be mapped to virtual memory space in units of *pseudo-superpages* (e.g., 2MB). Only one TLB entry is required for a large region of pseudo-memory space, which alleviates conflicts in the TLB — the TLB is now filled with virtual-to-pseudo translations.

Another mapping function is required between pseudo-physical memory and real physical memory. The OS can perform coloring on pseudo-superpages, and the pseudo-to-physical mapping guarantees that a pseudo-superpage is mapped to specific cache sets. We discuss the mapping function in the following subsection.

3.2 Swap-based Superpage Coloring

Our swap-based scheme is shown in Figure 5. k bits of the pseudo-superpage number field are taken as color bits that classify pseudo-superpages into 2^k colors. The k color bits in the pseudo-superpage number field and another k bits from the pseudo-superpage offset field, referred to as *candidate bits*, are swapped to generate the real physical address. The selection of k candidate bits from the pseudo-superpage offset guarantees that the color bits are swapped into the cache set index of the real physical address.

[2]Color region is a contigous part of a superpage that maps into the same cache sets.

SWAP has the following properties:

1. *A swap is a simple one-to-one mapping, which simplifies both hardware and software.*

   The swap can be easily implemented in hardware with negligible cost, omitting any additional management of free space and translation related to real physical memory space either in software or hardware.

2. *A swap distributes a pseudo-superpage into particular regions of multiple physical superpages.*

   Given k swapped bits, all the addresses in a pseudo-superpage have the same color bits (in the superpage number field) and take all the 2^k values (0, 1, 2, ..., $2^k - 1$) of the candidate bits (in the superpage offset field). After swapping, all the new addresses have the same value in the k bits of the superpage offset field, and 2^k different values in the k bits of the superpage number field. Thus, all these physical addresses are mapped to particular regions in 2^k physical superpages. Each region is a contiguous physical space within a superpage.

3. *All the regions corresponding to one pseudo-superpage are mapped into the same cache sets with the specified color.*

   SWAP's mapping guarantees that all the regions have the same k bits (i.e., the color bits) in the cache set index, thus, all are mapped to the same cache sets. Each such region is called a color region. Figure 6 shows an example of swap-based mapping.

With the support of pseudo-physical memory and swap-based pseudo-to-physical mapping, the OS can selectively allocate pseudo-superpages with particular colors in the application's virtual memory, where each color maps to specific cache sets.

3.3 Design Choices

The design choices include where and how the bit-swap is implemented.

The bit-swap can be implemented just before the access of the shared LLC. In this case, the private caches are addressed in pseudo-physical memory space, and only the LLC can be colored. Or, one can implement the bit-swap before

any level of private caches if page coloring in the private caches is useful. For example, in simultaneous multithreading (SMT), the private caches are shared by multiple hardware threads inside a core, and performing cache partitioning will eliminate contention between threads.

The bit-swap can be statically implemented in hardware, which is simple and incurs negligible hardware overhead. Or, the number of swapped bits and which bits to swap can be configurable for flexibility. For example, using different candidate bits, one can perform partitioning to memory channels [19] and banks [12, 16]. The latter methods may incur extra latency to perform swapping.

4. EVALUATION METHODOLOGY

We briefly explain our tools and workloads.

4.1 Simulation Setup

We use MARSSx86 [21], a full system cycle-accurate simulator, to model a four-core x86 processor. The system configuration is shown in Table 1. We modify the Linux kernel 2.6.32 to support page coloring on superpages.

For each simulation, we first fast-forward 10 billion instructions, and then simulate 1 billion instructions. For each workload, we run the simulation for five times, and report the average results excluding the maximum and minimum.

Table 1: System Configuration

CPU	4 OoO cores, 2.5 GHz, 4-wide issue, 128-entry ROB
Cache	L1-I Cache: 32 KB, private, 8-way, 4-cycle hit latency L1-D Cache: 32 KB, private, 8-way, 4-cycle hit latency L2 Cache: 256 KB, private, 8-way, 11-cycle hit latency L3 Cache: 8 MB, shared, 16-way, 28-cycle hit latency
TLB	ITLB for 4 KB page: 64-entry, private, 4-way DTLB for 4 KB page: 64-entry, private, 4-way DTLB for 2 MB page: 32-entry, private, 4-way STLB for 4 KB page: 512-entry, private, 4-way [a] 7-cycle penalty for DTLB miss and STLB hit [b]
Memory	Simple main memory model, 16GB 2 channels, 2 ranks per channel, 130-cycle latency

[a] STLB: Secondary TLB, only for 4 KB pages.
[b] DTLB hit latency is overlapped with L1 access latency.

4.2 Workloads

Since we focus on the performance of the memory system, i.e., TLB and LLC, we choose 13 memory intensive benchmarks from the SPEC CPU2006 benchmark suite [10] — those for which the LLC Misses Per Thousand Instructions (MPKI) is greater than 1.0 — and use reference input sizes. We randomly generate eight multi-programmed workloads from the 13 benchmarks. The details of each workload and coloring policy are given in Table 2. Note that the coloring policies we adopt are intuitive and not optimal. Achieving the optimal cache partitioning policy is beyond the scope of this paper.

4.3 Metrics

We use *weighted speedup* [25] to measure the system performance of the multi-programmed workloads. We normalize each benchmark Instruction Per Cycle (IPC) to that of the solo execution and calculate weighted speedup as the sum of all normalized benchmark IPCs. Since we simulate for a constant number of instructions, we use the total number of TLB misses and LLC misses of all benchmarks in a workload to evaluate the behavior of TLB and LLC.

Table 2: Multi-programmed Workloads and Coloring Policy

Workload	Combination	Coloring Policy[1]
WL1	429.mcf, 470.lbm, 462.libquantum, 459.GemsFDTD	8,4,2,2
WL2	471.omnetpp, 403.gcc, 462.libquantum, 459.GemsFDTD	
WL3	471.omnetpp, 482.sphinx3, 459.GemsFDTD, 433.milc	6,6,2,2
WL4	429.mcf, 482.sphinx3, 462.libquantum, 433.milc	
WL5	429.mcf, 436.cactusADM, 437.leslie3d, 462.libquantum	6,4,4,2
WL6	471.omnetpp, 470.lbm, 403.gcc, 462.libquantum	
WL7	403.gcc, 400.perlbench, 450.soplex, 470.lbm	4,4,4,4
WL8	470.lbm, 434.zeusmp, 403.gcc, 450.soplex	

[1] Specifies the number of colors assigned to each program respectively. The color bits are bit 15-18, so there are 16 colors.

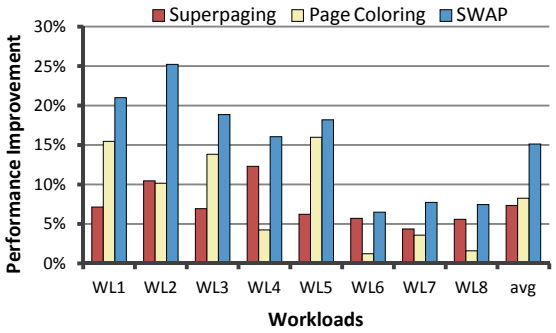

Figure 7: Performance improvement of various mechanisms compared to the baseline.

5. RESULTS

We evaluate the following mechanisms:

- **Baseline**: 4KB page, without cache partitioning.
- **Superpaging**: 2MB page, without cache partitioning.
- **Page Coloring**: 4KB page, with cache partitioning.
- **SWAP**: 2MB page, with cache partitioning.

5.1 Performance

Figure 7 shows the system performance (weighted speedup) improvement of various mechanisms compared to the baseline mechanism. All three optimizations — *superpaging, page coloring, and SWAP* — improve performance by an average of 7.34% (up to 12.3%), 8.26% (up to 16.0%) and 15.1% (up to 25.2%), respectively.

SWAP has the highest performance improvement, which demonstrates that superpaging and page coloring are orthogonal, and we have successfully bridged the gap between them by integrating the two optimizations through our proposed SWAP mechanism.

For superpage optimizations, workloads with larger memory footprints and random access patterns will improve more. For example, WL4 contains 429.mcf and 433.milc, which have footprints of 1.6GB and 679MB, respectively. Note that most SPEC CPU2006 benchmarks have small footprints, even with the reference input size. For larger applications, the performance improvement of superpage can be more significant [18].

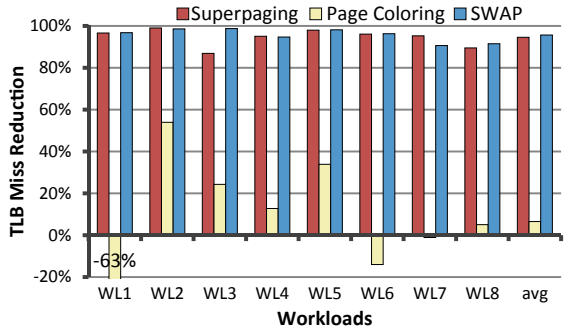

Figure 8: TLB miss reduction of various mechanisms compared to the baseline.

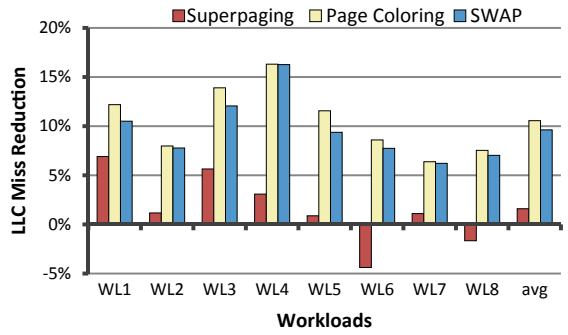

Figure 9: LLC miss reduction of various mechanisms compared to the baseline.

For page coloring, workloads with non-uniform cache partitions tend to see higher performance improvements. For instance, WL1 and WL2 assign one eighth colors to one of the four programs. These workloads usually contain programs that are insensitive to LLC capacity but hurt the performance of the other co-running programs. For example, 462.libquantum is a streaming program, so if the LLC is shared without partitioning, this program will continually flush the LLC and invalidate the other co-running programs' useful data. Besides, the performance of 462.libquantum is insensitive to the LLC capacity, and so assigning less LLC capacity (fewer colors) to such programs will not degrade their performance but will free LLC capacity for other programs, which improves the system throughput.

5.2 TLB and LLC Behavior

Figure 8 and Figure 9 show the reduction of total TLB misses and LLC misses for different mechanisms compared to the baseline.

As shown in Figure 8, both superpaging and our proposed SWAP mechanism can significantly reduce TLB misses by an average of 94.6% (up to 99.0%) and 95.7% (up to 98.8%), respectively. The page coloring scheme has an unpredictable effects on TLB misses. For example, when using page coloring, WL2 reduces TLB misses by 54.0%, while WL1 increases TLB misses by up to 62.7%.

As shown in Figure 9, page coloring can reduce LLC misses by an average of 10.6% (up to 16.3%). SWAP performs similarly to page coloring, with an average reduction of 9.62% (up to 12.1%). Superpaging also has unpredictable effects on LLC misses.

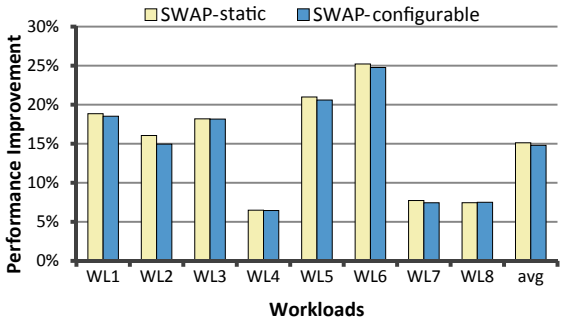

Figure 10: Performance degradation to make swap configurable.

In summary, SWAP can simultaneously reduce TLB misses and LLC misses, leading to better performance than either superpaging or page coloring alone.

5.3 Static vs. Configurable Swapping

To make the bit-swap operation configurable for flexibility, extra latency may be introduced. Since we only perform cache partitioning on the LLC, the configurable bit-swap can be done between the private cache and the shared LLC. This adds an extra cycle latency for LLC accesses, e.g., the 29-cycle L3 hit latency in our experimental platform. We evaluate its impact on performance, results for which are shown in Figure 10. The average improvement compared to the baseline for the configurable swapping scheme is 14.8%, slightly lower than that of static swapping scheme (15.1%). This result suggests that we can maintain flexibility in the architecture design with negligible performance impact.

6. RELATED WORK

Superpaging: Superpaging is a straightforward method to reduce TLB conflicts by increasing TLB reach. Almost all processor architectures support superpages. Talluri et al. discuss the tradeoffs of supporting multiple page sizes in hardware [27]. Superpaging needs the support of the OS [26, 9, 20, 2, 1]. Roemer et al. [23] and Fang et al. [8] study dynamic superpage promotion under different kinds of memory system support.

Page Coloring: Page coloring is a classic technique that has been extensively used for code and data placement in order to reduce cache misses [29, 13, 4, 24]. In recent years, page coloring has been used to partition shared resources such as caches [28, 14, 30, 15, 17, 7], memory channels [19], and banks [12, 16] to address the severe contention caused by sharing. Nevertheless, none of these efforts recognize the implementation contradiction that prevents simultaneously leveraging superpaging and page coloring. Our work is orthogonality to them — by swapping particular bits to overlap with the cache set index, channel index, or bank index, we can partition of caches, channels or banks while using superpaging.

Scattered superpage: As previously mentioned, scattered superpage [6] adds extra latency to TLB lookup due to the additional calculation, and it requires additional OS effort to manage the virtual-to-physical mapping (besides the OS page coloring). Our hardware modification is a simple bit-swap with negligible overhead. And our approach

only requires the traditional OS page coloring implementation, which will classify pseudo-superpages into 2^k free-lists, and allocate pseudo-superpages with specific colors when demanded. The hardware one-to-one bit-swap maps pseudo-superages into corresponding color regions in physical superpages, omitting any additional management or table-based translation.

7. CONCLUSION

In this paper, we propose SWAP, a swap-based cache set index scheme. SWAP is a simple, low-cost method to leverage both superpaging and page coloring optimizations simultaneously. SWAP decouples the direct mapping between virtual superpages to physical superpages by introducing a pseudo-physical memory space. A bit-swap operation that translates a pseudo-physical address to a real physical address ensures that a pseudo-superpage is mapped to particular cache sets. Our evaluation shows that SWAP improves performance by 15.1% on average (up to 25.2%) by reducing of TLB and LLC misses by 95.7% and 9.62%, respectively.

8. ACKNOWLEDGMENTS

We are grateful to Sally A. McKee for helping to improve this paper. We thank the anonymous reviewers for their feedback. This work is supported by the National Natural Science Foundation of China (NSFC) under the grant number 61221062, 61272132, and 61331008, the National Basic Research Program of China (973 Program) under the grant number 2011CB302502, the Strategic Priority Research Program of the Chinese Academy of Sciences under the grant number XDA06010401, and Huawei Research Program under the grant number YBCB2011030. Yungang Bao is partially supported by the CCF-Intel Young Faculty Research Program (YFRP) Grant.

9. REFERENCES

[1] Huge pages/libhugetlbfs, 2010. http://lwn.net/Articles/374424/.

[2] A. Arcangeli. Transparent hugepage support. In *KVM Forum*, 2010.

[3] R. Bhargava, B. Serebrin, F. Spadini, and S. Manne. Accelerating two-dimensional page walks for virtualized systems. In *ASPLOS*, 2008.

[4] E. Bugnion, J. M. Anderson, T. C. Mowry, M. Rosenblum, and M. S. Lam. Compiler-directed page coloring for multiprocessors. In *ASPLOS*, 1996.

[5] D. Chandra, F. Guo, S. Kim, and Y. Solihin. Predicting inter-thread cache contention on a chip multi-processor architecture. In *HPCA*, 2005.

[6] 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 *ICCD*, 2013.

[7] X. Ding, K. Wang, and X. Zhang. ULCC: a user-level facility for optimizing shared cache performance on multicores. In *PPoPP*, 2011.

[8] Z. Fang, L. Zhang, J. B. Carter, W. C. Hsieh, and S. A. McKee. Reevaluating online superpage promotion with hardware support. In *HPCA*, 2001.

[9] N. Ganapathy and C. Schimmel. General purpose operating system support for multiple page sizes. In *USENIX ATC*, 1998.

[10] J. L. Henning. SPEC CPU2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 2006.

[11] Intel. Memory cache control. In *Intel 64 and IA-32 Architectures Software Developer's Manual - Volume 3: System Programming Guide*, 2012.

[12] M. K. Jeong, D. H. Yoon, D. Sunwoo, M. Sullivan, I. Lee, and M. Erez. Balancing dram locality and parallelism in shared memory cmp systems. In *HPCA*, 2012.

[13] R. E. Kessler and M. D. Hill. Page placement algorithms for large real-indexed caches. *ACM TOCS*, 1992.

[14] J. Lin, Q. Lu, X. Ding, Z. Zhang, X. Zhang, and P. Sadayappan. Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems. In *HPCA*, 2008.

[15] J. Lin, Q. Lu, X. Ding, Z. Zhang, X. Zhang, and P. Sadayappan. Enabling software management for multicore caches with a lightweight hardware support. In *SC*, 2009.

[16] L. Liu, Z. Cui, M. Xing, Y. Bao, M. Chen, and C. Wu. A software memory partition approach for eliminating bank-level interference in multicore systems. In *PACT*, 2012.

[17] Q. Lu, J. Lin, X. Ding, Z. Zhang, X. Zhang, and P. Sadayappan. Soft-OLP: Improving hardware cache performance through software-controlled object-level partitioning. In *PACT*, 2009.

[18] C. McCurdy, A. L. Coxa, and J. Vetter. Investigating the TLB behavior of high-end scientific applications on commodity microprocessors. In *ISPASS*, 2008.

[19] S. P. Muralidhara, L. Subramanian, O. Mutlu, M. Kandemir, and T. Moscibroda. Reducing memory interference in multicore systems via application-aware memory channel partitioning. In *MICRO*, 2011.

[20] J. Navarro, S. Iyer, P. Druschel, and A. Cox. Practical, transparent operating system support for superpages. In *OSDI*, 2002.

[21] A. Patel, F. Afram, S. Chen, and K. Ghose. MARSSx86: A Full System Simulator for x86 CPUs. In *DAC*, 2011.

[22] M. K. Qureshi and Y. N. Patt. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *MICRO*, 2006.

[23] T. H. Romer, W. H. Ohlrich, A. R. Karlin, and B. Bershad. Reducing tlb and memory overhead using online superpage promotion. In *ISCA*, 1995.

[24] T. Sherwood, B. Calder, and J. Emer. Reducing cache misses using hardware and software page placement. In *ICS*, 1999.

[25] A. Snavely and D. M. Tullsen. Symbiotic jobscheduling for a simultaneous mutlithreading processor. *ACM SIGPLAN Notices*, 2000.

[26] M. Talluri and M. D. Hill. Surpassing the tlb performance of superpages with less operating system support. In *ASPLOS*, 1994.

[27] M. Talluri, S. Kong, M. D. Hill, and D. A. Patterson. Tradeoffs in supporting two page sizes. In *ISCA*, 1992.

[28] D. Tam, R. Azimi, L. Soares, and M. Stumm. Managing shared l2 caches on multicore systems in software. In *WIOSCA*, 2007.

[29] G. Taylor, P. Davies, and M. Farmwald. The tlb slice - a low-cost high-speed address translation mechanism. In *ISCA*, 1990.

[30] X. Zhang, S. Dwarkadas, and K. Shen. Towards practical page coloring-based multicore cache management. In *EuroSys*, 2009.