Page Classifier and Placer: A Scheme of Managing Hybrid Caches

Xin Yu, Xuanhua Shi, Hai Jin, Xiaofei Liao, Song Wu, Xiaoming Li

Page Classifier and Placer: A Scheme of Managing Hybrid Caches

Xin Yu[1], Xuanhua Shi[1], Hai Jin[1], Xiaofei Liao[1], Song Wu[1], Xiaoming Li[2]

[1]Services Computing Technology and System Lab
Cluster and Grid Computing Lab
School of Computer Science and Technology
Huazhong University of Science and Technology,Wuhan, 430074, China
xhshi@hust.edu.cn
[2]Department of ECE, University of Delaware, Newark, DE, USA

Abstract. Hybrid cache architecture (HCA), which uses two or more cache hierarchy designs in a processor, may outperform traditional cache architectures because no single memory technology can deliver the optimal power, performance and density at the same time. The general HCA scheme has also been proposed to manage cache regions that have different usage patterns. However previous HCA management schemes control data placement at cache set level and are oblivious to software's different power and performance characteristics in different hardware cache regions. This hardware-only approach may lead to performance loss and may fail to guarantee quality of service. We propose a new HCA approach that enables OS to be aware of underlying hybrid cache architecture and to control data placement, at OS page level, onto difference cache regions. Our approach employs a light-weighted hardware profiler to monitor cache behaviors at OS page level and to capture the hot pages. With this knowledge, OS will be able to dynamically select different cache placement policies to optimize placement of data to achieve higher performance, lower power consumption and better quality of service. Our simulation experiments demonstrate that the proposed hybrid HCA achieves 7.8% performance improvement on a dual-core system compared to a traditional SRAM-only cache architecture and at the same time reduces area cost.

Keywords: hybrid cache, page coloring, multi-core.

1 Introduction

Cache is widely used in todays computers to mend the ever-increasing speed gap between processor core and main memory. Emerging memory technologies have demonstrated significantly different properties in density, speed, power consumption, reliability features, and scalability. Table 1 summarizes the important characteristics of four memory technologies: SRAM, Phase-change RAM (PRAM) [4, 6], embedded Dynamic RAM (eDRAM), and Magnetic RAM (MRAM) [5].

Table 1. Characteristic comparison of different memory technologies

Features	SRAM	eDRAM	MRAM	PRAM
Density	Low	High	High	Very high
Speed	Very fast	Fast	Fast read; Slow write	Slow read; Very slow write
Dynamic Power	Low	Medium	Low read; High write	Medium read; High write
Leak Power	High	Medium	Low	Low
Non-volatile	NO	NO	Yes	Yes
Scalability	Yes	Yes	Yes	Yes

Hybrid Cache Architecture (HCA) has been proposed to take advantage of multiple memory technologies [4–6] in one cache. However, the existing HCA management schemes control data placement at cache set level and hide from software the knowledge about differences in power and performance characteristics of hardware cache regions. This hardware-only approach may lead to performance loss and loss of quality of service.

To address the shortcomings of existing HCA management schemes, we propose a new approach which makes operating system (OS) be aware of the underlying HCA architecture and enables OS to customize data placement in HCA and focus on using our proposed HCA technology to improve the throughput of multicore systems.

This paper makes the following contributions:

- We extend the page coloring capability in OS with a novel awareness of L2 cache access patterns and program behavior. In particular, our technique for the first time dynamically manages hybrid cache at page level through page migration and optimize migration policy to amortize the performance overhead.
- We propose a hardware-assisted mechanism page classifier to monitor the patterns of L2 cache accesses from each core at page granularity. The page classifier could not only monitor L2 accesses but also capture hot pages.
- We propose an effective heuristic to decide when and how to migrate hot pages in or out of fast regions, so as to make full use of HCAs large capability and high access speed at the same time.

2 Proposed Scheme

We assume the shared L2 is divided into a fast region and a slow region. The heterogeneous cache placement is only possible if we can monitor the access frequency of cache blocks and dynamically adjust placement of cache blocks. We propose a hardware-assist software-controlled hybrid mechanism to address the two challenges.

2.1 Page Coloring

Traditionally, the intersection of page number bits and L2 index bits are used as the page color bits, as shown in Fig.1. OS has control over these bits. We choose certain subset of those bits to identify different cache regions, which is called hybrid bits. Thus, when a page is migrated to specified page with certain hybrid bits, its data would be accessed in that cache region. In this study, we assume the size of a page is 4KB and the size of L2 is 128KB. Consequently there are six page color bits in the cache subsystem used by this work, as shown in Fig.1. As the ratio of fast region to slow region is 1:3, the first two bits of page color bits are referred to hybrid bits.

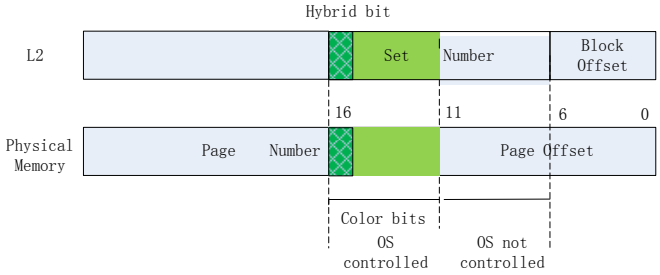


Fig. 1. Page coloring bits and hybrid bits

2.2 Page Classifier

In this study, we enhance L2 cache with a new module: a page classifier. The page classifier is composed of two parts, named Sampler and Mature/Nursery. Mature/Nursery counts the number of L2 access of every page. To reduce the power overhead of the added components, a L2 access filter Sampler is designed and controls how often L2 accesses are counted.

Mature/Nursery classifies the L2 access pattern at page level. They are counter caches. Every block records the access behavior of one page. Its data could include read counter, write counter and total access counter. As it is indexed via physical page number, its tag is the segment of page number.

As what they are called, pages access pattern would be initially recorded in Nursery. When one page becomes relatively hot, its block would be exchanged into Mature. To archive this purpose, Mature and Nursery are designed to have the same cache sets. Block swap happens between the same cache set of Mature and Nursery. As a result, a new L2 access is firstly recorded in Nursery. When the page has more and more L2 accesses, which is higher than other blocks, it would be swapped into Mature. We set a swapping frequency for this progress to avoid block jitters. The access to L2 cache and the access to Mature/nursery are parallel. If the access is a L1 cache miss, the physical address is sent to L2 cache module. If the access to Mature/Nursery is a hit, the read or write counter is increased by one. If it is a miss, the block with least number of accesses (LATBlock) in Nursery would be replaced and the counter would be reset to one.

We exchange blocks between Mature and Nursery periodically. OS compares the blocks of biggest number of accesses in Nursery, named MATBlock, with LATBlock in Mature. If the former is bigger, swapping would happen. After swapping, MAT in Nursery and LAT in Mature both need to update.

2.3 Page Placer

The page placer is designed to determine where and when to migrate a candidate page to a new physical page. Os would scan Mature to find hot pages still in slow region and scan Nursery to find not-hot pages still in fast region periodically.

When choosing page migration destination, we need to specify its page color. We take a round-robin policy in allocating physical pages, to make virtual pages distributed evenly among different colors in memory. This round-robin process of assigning page number is not only used by the page migration process, but also adopted in the allocation of physical pages for virtual pages. In this study, two registers are provided to assist this progress, which record next page color (NPC) for each region. After one page is allocated, the value of next page color would be increased by one; when it reaches the last page color of one region, it would reset to zero. To adapt this policy, traditional buddy system algorithm for managing free page frames is slightly modified: choosing appropriate bulks of page frames for migration destination. For example, we assume 27 pages need to be migrated to fast region. OS would search the list of blocks for buddy system to find groups of 32 contiguous page frames. It would start to allocate page from the page color recorded in NPC register of fast region and allocate 27 pages. After page allocation, the register would add 27 accordingly.

To reduce page migration cost, one simple yet effective approach is to decide the migration time separately for hot pages and not-hot pages. For hot pages, page fault is triggered and the page would be copied to its destination page at once, and thus its blocks are updated to fast region of L2 immediately. As most part of its data has already been in L2, the copy process costs little time. However, not-hot pages is to keep in a drowsy mode: they are invalided and written back into swapping area of the disk in migration period; only when its data is accessed, it writes into its destination page frame. Thus its data would not pollute fast region because they avoid being swapped in and out of L2 frequently. As a result, those policies make page migration damage the performance of the applications slightly.

To reduce migration cost further, not only migration frequency is controlled, but also hot page threshold (HotThreshold) is set. As the analysis above, pages in Mature, which relatively hotter than pages of the same set in Nursery are unnecessarily globally hottest. Therefore HotThreshold is set. There is a formula to describe the relationship of system parameters and estimate an approximate yet reasonable value for HotThreshold:

$$HotThreshold = \frac{MigrCycle \times IPC \times Ratio}{CapabilityMN} \times Multiplier \tag{1}$$

In Eq. 1, *IPC* is the number of instructions per cycle of the running programs, *MigrCycle* is the page migration cycle, *CapabilityMN* is the total entry number of Mature and Nursery, *Ratio* represents the average times of L2 cache access per instruction, and Multipliecr represents the ratio of HotThredhold to the average times of L2 accesses per page. In this formula, HotThreshold varies directly with *MigrCycle*. As *IPC* and *Ratio* are the characteristics of the programs, we only change *Multiplier* to adjust the ratio of hot pages and set it as 10 in this work. In order to dynamically reflect the L2 access pattern of one page, all counters in Mature/Nursery are aged by right shift by one bit at the end of migration period.

3 Methodology

In this section, we describe our simulation methodology.

3.1 System Configuration

We choose the simulation parameters based on the related studies [4–6,9,12], and we use the typical density, latency, and energy numbers for the three memory technologies, which are calculated using CACTI 6.0 [8]. We scale these parameters to 65nm technology as described in [1]. We use the same cache parameters as described in [11], which are shown in Table 2.

Table 2. Four memory technology parameters

Cache	Normal Density	Latency (cycles)	Dynamic energy (nJ)	Static power (W)
SRAM (1MB)	1	8	0.388	1.36
eDRAM (4MB)	4	24	0.72	0.4

We choose Zesto [7] as our base simulator, which is a cycle-level x86 processor simulator publicly available for academic use. We augment its cache part to study the proposed hybrid cache management scheme. Our system configuration is summarized in Table 3.

Table 3. System configuration

Processor	3000MHz, out of order, (8 way issues), core number depends on design
L1	32KB DL1, 32KB IL2, 64B8way, 8bank, (1 R/W port)
L2 (LLC)	shared LLC64B, 16way, 16 bank, latency and capability depends on design
Memory	400 cycle latency, (memory contr. vs. core speed 1:2),page size:4KB

3.2 Workload

We choose SPEC CPU 2006 as the benchmarks to run on the simulated system. In order to run on multi-core simulated by Zesto, we use a program, *zesto-eio*,

provided by Zesto to generate *eio* files and we got 21 *eio* files successfully out of total 29 benchmarks shown in Table 4. After a warm-up period simulation of 100 million instructions, we simulate the system cycle-by-cycle for 100 million instructions and collect the simulation results.

Table 4. Workloads

Benchmarks	Applications
Spec CINT2006	400.perlbench, 401.bzip2, 403.gcc, 429.mcf, 445.gobmk, 456.hmmer, 458.sjeng, 462.libquantum, 464.h264ref Spec
CFP2006	410.bwaves, 433.milc, 434.zeusmp, 435.gromacs, 436.cactusADM, 437.leslie3d, 444.namd, 447.dealII, 450.soplex, 453.povray, 465.tonto, 470.lbm

3.3 Design Methodology

To take advantage of separate characteristics of different memory technologies, we present the hybrid cache subsystem. To compare the performance of the hybrid cache scenario and pure-SRAM cache scenario, we assume that the chip area is the same for all the design cases.

Before we introduce the design methodology, we define the division of tasks between the hardware part (the page classifier) and the software part (the page placer in OS). The hardware part is responsible for profile cache access behaviors of programs. It records how many times a virtual page accesses the L2 and filters the hot page. In this process it does not care the L2 architecture, no matter it is homogeneous like pure-SRAM cache, or it is heterogeneous, such as hybrid cache consisted of SRAM and eDRAM, or it is consisted of SRAM and MRAM, even consisted of eDRAM and MRAM. This means that the design of hardware part is not affected by L2 architecture. OS is the only part that should be aware of L2 architecture. Thus it could re-adjust which part of L2 could be accessed by one virtual page, according to the attributes of different RAM technologies. By simply configuring and taking advantage of those attributes in the page placer algorithm, OS could optimize performance of L2 architecture. As analyzed above, the hardware part is aware of software behaviors and the software part is aware of the hardwares attribution. The unique combination of the two-way awareness enables OS to control the behaviors of cache access without complex hardware design.

Therefore, this design is almost agnostic to the design of HCA (hybrid cache architecture), which makes it scalable for different HCAs. The porting to different HCAs is merely to change the configuration in the page placer algorithm. To improve scalability, this design adopts a simple but efficient way and demonstrates that this hardware-software combined design could work very well for HCA. The advantage can be fully illustrated with a small scale system: dual core and 2MB L2 on CMP, rather than on prevalent larger scale systems.

The design of HCA also follows this simple but efficient methodology. The quickest SRAM and slowest eDRAM are used, to illustrate that even in such radical combinations, our technique can perform well.

There are clear benefits of such hybrid cache design: (1) The new memory technology has a much higher density than traditional SRAM technology, which increases the effective cache size under the same chip area constraint. (2) Performance can be improved by keeping hot cache lines which are accessed relatively most frequently in fast regions and place not so hot cache lines in slow regions. (3) This hardware-software combined design has simplified the process of making OS aware of L2 behavior and controlling it, and is scalable for different HCAs. (4) Flexible and tunable page placing strategies become possible and promising.

4 Result

In this section, we present experimental results of HCA.

4.1 General Evaluation

Table 5. Four sets of cache L2 parameters

	L2 parameters
Conf.1	512KB SRAM-only (8 cycles)
Conf.2	1MB SRAM-only (8 cycles)
Conf.3	Fast region: 512KB SRAM (8 cycles); Slow region: 1.5MB eDRAM (24 cycles)

We assume the total size of the hybrid cache is 2MB, and the size ratio of SRAM and eDRAM is 1: 3. Under the same area constraint, we should study 0.875 MB SRAM as control set. To avoid complicated indexing schemes which are often associated with odd-sized caches, we construct 0.5 MB and 1MB SRAM instead to approach the performance of 0.875 MB SRAM , shown as *Conf.1* and *Conf.2*.

4.2 Results of Page Classifier

To check whether page classifier can pick out the hot pages effectively, we analyze the ratio of the access number of identified hot pages to the total L2 access number. As shown in Fig.2, the ratio is over 0.8 for more than half of benchmarks, and the average rate is over 0.6. The results proves that 1) majority of L2 cache accesses are belong to very small set of pages, and 2) the page classifier does a good job to identify these hot pages.

4.3 Results on Single-Core

First, we apply the page placer to HCA. We name HCA architecture with OS Cache Management as HCACM and HCA refers to the common HCA without

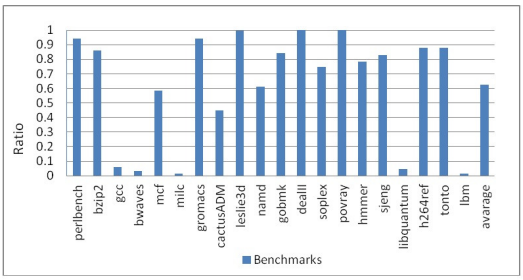
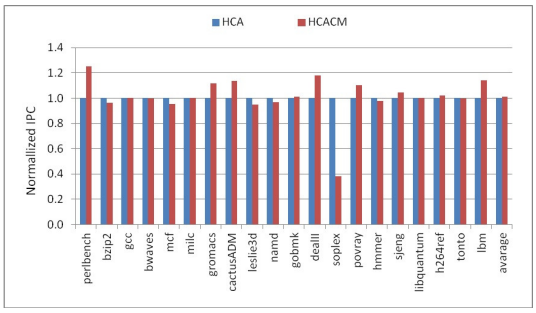


Fig. 2. Ratio of hot page L2 access number to total L2 access number

OS cache management. Fig.3 compares the performance of HCACM and HCA, and shows that the average IPC improvement is only 1%.



0

Fig. 3. Comparison of normalized IPC between HCA enabled cache management and not enabled cache management.

To find out why this page placer does not work well, we compare the L2 miss rate between HCA and HCACM in Fig.4. We can see that the benchmarks can be divided into three categories: 1) the hot pages have high access frequency and the page placer improves the L2 access performance, such as *perlbench*, *gromacs*, *cactusADM*; 2) the hot pages have high access frequency, but their performance almost stays the same, or even degrades, such as *mcf*, *namd*, *soplex*, and *hmmer*; 3) page classifier is not so useful for them, the hot page access frequency is lower than normal and the performance nearly stays the same, for example *gcc*, *bwaves*, *milc*, *libquatum*.

The analysis of the results above is consistent with the benchmarks L2 access behavior. The first category of benchmarks are memory-latency sensitive and has small working set, therefore benefiting more from our policy that hot pages are all placed in fast region. The second category of benchmarks has a larger working set that cannot wholly fit in to fast region. Therefore the total L2 miss rate increases much when putting all hot pages in fast region, which could offset performance improvement from low latency, or even hurts the performance. The last category of benchmarks is non-memory sensitive, so they almost stay the same no matter we apply cache management or not.

To make page placer perform better in HCA, we focus on the first two categories of benchmarks and pursue a better policy to place hot pages. To resolve

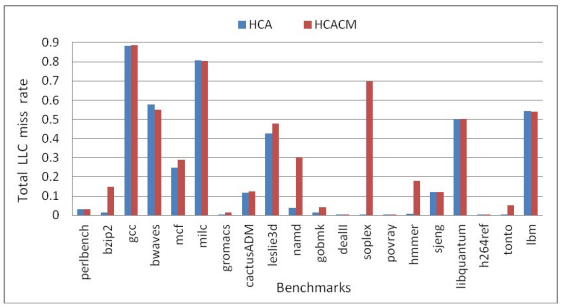


Fig. 4. Comparison of total L2 miss rate between HCA and HCACM.

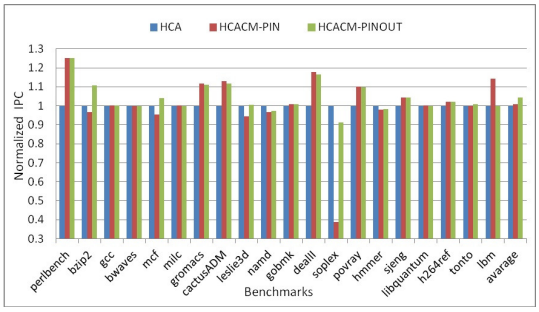


Fig. 5. Comparison of normalized IPC between HCA, HCACM with migration policy PIN and HCACM with migration policy PINOUT.

the problem, the page placer not only migrates hot pages into fast region but also migrates some out of fast region when its L2 miss rate is high in fast region than slow region by 10% margin. To distinguish between the new policy and the former one, we call the new policy PINOUT and the former one PIN. We compare the two policies in HCA, and the results are shown in Fig.5.

In Fig.5, we can see the following: 1) the average performance improvement for policy PINOUT for all the 20 benchmarks is about 7.6% over policy PIN, 4.2% over HCA; 2) The performance of the second category of benchmarks with policy PINPOUT increases 27.7% over policy PIN. But the performance of the first category decreases slightly, because they have small working set and do not benefit from large capability.

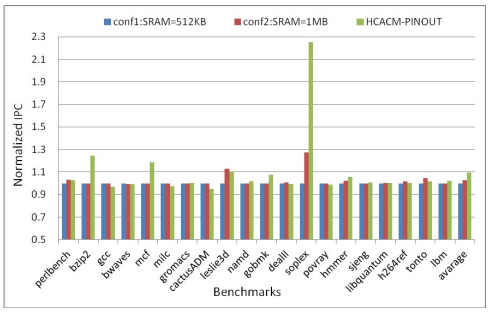


Fig. 6. Comparison of normalized IPC between *Conf.1*, *Conf.2* and HCACM with migration policy PINOUT.

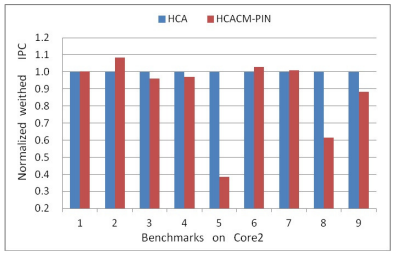
We compare the HCACM-PINOUT (HCACM with migration policy PINOUT) with SRAM with two configurations (*Conf.1* and *Conf.2* in Table 5), and the results are shown in Fig.6. We can see that the average IPC of HCACM-PINOUT is 9.5% higher than that of 512 KB SRAM and 5.8% higher than that of 1MB SRAM. Especially for *bzip2*, *mcf*, and *soplex*, the IPC is more than 10% higher than 512KB SRAM. It is worth noting that although cache area used by HCACM-PINOUT is smaller than that of a 1MB SRAM L2 cache, it performs much better than 1MB SRAM L2 cache. This confirms that not only short latency but also large capability bring good performance for HCA when reasonable management is applied.

4.4 Results on Dual-Core

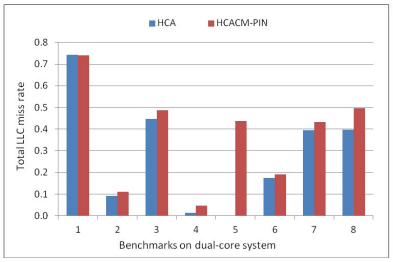
We also apply this mechanism to dual-core system, and we run eight sets of benchmarks, shown in Table 6, on dual-core system with different L2 cache configurations and management policies.

Table 6. Eight sets of benchmarks on dual-core system

Num.	Benchmarks	Num.	Benchmarks
1	gcc+bwaves	5	povray+hmmer
2	gromacs+cactusADM	6	sjeng+libquantum
3	leslie3d+namd	7	h264ref+tonto
4	gobmk+dealII	8	lbm+soplex



(a) Normalized weighted IPC.



(b) Total L2 miss rate.

Fig. 7. sets of benchmarks running on dual-core system

We first run those benchmark sets on dual-core system with policy PIN. The IPC results illustrate that the overall performance decrease a little as shown in Fig.7(a) (the numbers of 1-8 refer to the sets of benchmarks, and 9 refers to the average IPC of all the sets).

The PIN policy on dual cores has similar problem with that on single core, which is proven by the comparing the total L2 miss rate between HCA and HCA-PIN in Fig.7(b). Multiprogram has aggravated the competition in L2 and lead to high cache jitter except set 2 and 6, which are more latency-insensitive than capacity-insensitive.

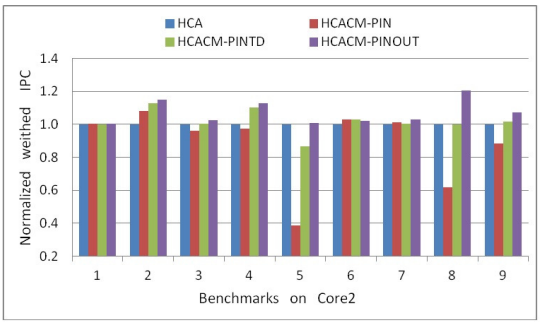


Fig. 8. Normalized weighted IPC of sets of benchmarks running on dual-core system.

To solve the cache jitter, we constrain that a candidate hot page could migrate only if its miss rate is lower than a specified miss rate threshold. We call this policy PINTD and run a series of experiments with it. We compare the IPCs for HCA, HCACM-PIN, HCACM-PINTD, and HCACM-PINOUT, and get the results in Fig.8. The HCACM-PINOUT refers to the results that we apply policy PINOUT based on PINTD. From Fig.8, we can see the followings: 1) the average performance improvement of HCACM-PINTD is just about 1.6% compared with PIN; 2) The average weighted performance improvement for PINOUT is 13.5% over PIN, 7.2% over HCA, 14.9% over *Conf.1*, 7.8% over *Conf.2*. Sets with higher cache miss rate could benefit more from HCACM-PINOUT. For 2, 4 and 8, they achieve over 10% improvement compared to HCA, especially for set 8 with about 21% improvement. Although set 5 is not suitable for HCA cache management, PINOUT could eliminate the bad effects of cache management. Above all, the page placer which works with policy PINOUT could handle HCA cache management well.

5 Related Work

In recent years, substantial research effort has been dedicated to intelligently manage hybrid cache at fine granularity.

There are many different hybrid cache studies for CMPs. Sun et al. [10] propose two architectural techniques: read-preemptive write buffer and SRAM-MRAM hybrid L2 cache to mitigate the long latency and high energy of MRAM when writing. Wu et al. [11] , discuss and evaluate two types of hybrid cache architectures, and propose HCA management scheme to control data placement at cache set level and hide from software the differences in power and performance characteristics of hardware cache regions.

Managing shared cache in CMPs, at both finer and coarser granularity has been widely studied [3], but few are applied to hybrid cache management. This method requires complex search to per-core private tag arrays that must be kept coherent, which adds extra design and hardware cost and some performance lost.To avoid the aforementioned problems, many other works manage shared caches in CMPs at page granularity. Chaudhuri et al. [2] have devised OS support

mechanism to allow page placement policies in NUMA systems. Awasthi et al [1] extend that concept with new mechanisms that allow the hardware and OS to dynamically manage cache capacity per thread as well as optimize placement of data shared by multiple threads.

In this work, we apply page coloring approach to overcome the shortcomings of existing HCA management schemes. Our key innovation is the introduction of a light-weighted hardware mechanism added to HCA to identify and collect cache behavior of hot OS pages.

6 Conclusion and Future Work

In this paper, we presented a hybrid cache architecture to construct on-chip cache hierarchies with different memory technologies. We proposed a light-weighted hardware mechanism to let OS be aware of underline hybrid cache architecture and studied page placer mechanism to control data placement onto difference cache regions at OS page level.

Overall, we showed the potential benefits of applying hybrid caches to improve the cache subsystem performance with OS-aware cache management. As an initial study, we have mainly presented page-level cache management. In future work, if the extra hardware mapping layer is employed, the granularity of classification and placement can be arbitrary.

7 Acknowledgments

This work is supported by the NSFC under grants No. 61370104 and No.61133008, National Science and Technology Pillar Program under grant 2012BAH14F02, MOE-Intel Special Research Fund of Information Technology under grant MOE-INTEL-2012-01, and Chinese Universities Scientific Fund under grant No. 2014TS008.

References

1. Awasthi, M., Sudan, K., Balasubramonian, R., Carter, J.: Dynamic hardware-assisted software-controlled page placement to manage capacity allocation and sharing within large caches. In: Proceedings of IEEE 15th International Symposium on High Performance Computer Architecture (HPCA'09), pp. 250–261. IEEE (2009)
2. Chaudhuri, M.: Pagenuca: Selected policies for page-grain locality management in large shared chip-multiprocessor caches. In: Proceedings of IEEE 15th International Symposium on High Performance Computer Architecture (HPCA'09), pp. 227–238. IEEE (2009)
3. Chishti, Z., Powell, M.D., Vijaykumar, T.: Distance associativity for high-performance energy-efficient non-uniform cache architectures. In: Proceedings of 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'03), pp. 55–66. IEEE (2003)

4. Hanzawa, S., Kitai, N., Osada, K., Kotabe, A., Matsui, Y., Matsuzaki, N., Takaura, N., Moniwa, M., Kawahara, T.: A 512kb embedded phase change memory with 416kb/s write throughput at 100μa cell write current. In: Proceedings of IEEE International Solid-State Circuits Conference (ISSCC'07), pp. 474–616. IEEE (2007)
5. Hosomi, M., Yamagishi, H., Yamamoto, T., Bessho, K., Higo, Y., Yamane, K., Yamada, H., Shoji, M., Hachino, H., Fukumoto, C., Nagao, H., Kano, H.: A novel nonvolatile memory with spin torque transfer magnetization switching: Spin-ram. In: Proceedings of IEEE International Electron Devices Meeting (IEDM'05), pp. 459–462. IEEE (2005)
6. Lam, C.: Cell design considerations for phase change memory as a universal memory. In: Proceedings of International Symposium on VLSI Technology, Systems and Applications (VLSI-TSA'08), pp. 132–133. IEEE (2008)
7. Loh, G.H., Subramaniam, S., Xie, Y.: Zesto: A cycle-level simulator for highly detailed microarchitecture exploration. In: Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'09), pp. 53–64. IEEE (2009)
8. Muralimanohar, N., Balasubramonian, R., Jouppi, N.P.: Cacti 6.0: A tool to model large caches. HP Laboratories (2009)
9. Pellizzer, F., Pirovano, A., Ottogalli, F., Magistretti, M., Scaravaggi, M., Zuliani, P., Tosi, M., Benvenuti, A., Besana, P., Cadeo, S., Marangon, T., Morandi, R., Piva, R., Spandre, A., Zonca, R., Modelli, A., Varesi, A., Lowrey, T., Lacaita, A., Casagrande, G., Cappelletti, P., Bez, R.: Novel μtrench phase-change memory cell for embedded and stand-alone non-volatile memory applications. In: Proceedings of International Symposium on VLSI Technology, Systems and Applications (VLSI-TSA'08), pp. 18–19. IEEE (2004)
10. Sun, G., Dong, X., Xie, Y., Li, J., Chen, Y.: A novel architecture of the 3d stacked mram l2 cache for cmps. In: Proceedings of IEEE 15th International Symposium on High Performance Computer Architecture (HPCA'09), pp. 239–249. IEEE (2009)
11. Wu, X., Li, J., Zhang, L., Speight, E., Rajamony, R., Xie, Y.: Hybrid cache architecture with disparate memory technologies. In: Proceedings of International Symposium on Computer architecture (ISCA'09), pp. 34–45. ACM (2009)
12. Zhao, W., Belhaire, E., Mistral, Q., Chappert, C., Javerliac, V., Dieny, B., Nicolle, E.: Macro-model of spin-transfer torque based magnetic tunnel junction device for hybrid magnetic-cmos design. In: Proceedings of the 2006 IEEE International Behavioral Modeling and Simulation Workshop (BMSW'06), pp. 40–43. IEEE (2006)