IEICE TRANS. INF. & SYST., VOL.E96–D, NO.12 DECEMBER 2013

PAPER Special Section on Parallel and Distributed Computing and Networking

Improving Cache Partitioning Algorithms for Pseudo-LRU Policies

Xi ZHANG[†a)], ***Nonmember***, **Chuanyi LIU**[†], ***Member***, **Zhenyu LIU**[††], ***Nonmember***, ***and*** **Dongsheng WANG**[††], ***Member***

SUMMARY As the number of concurrently running applications on the chip multiprocessors (CMPs) is increasing, efficient management of the shared last-level cache (LLC) is crucial to guarantee overall performance. Recent studies have shown that cache partitioning can provide benefits in throughput, fairness and quality of service. Most prior arts apply true Least Recently Used (LRU) as the underlying cache replacement policy and rely on its stack property to work properly. However, in commodity processors, pseudo-LRU policies without stack property are commonly used instead of LRU for their simplicity and low storage overhead. Therefore, this study sets out to understand whether LRU-based cache partitioning techniques can be applied to commodity processors. In this work, we instead propose a cache partitioning mechanism for two popular pseudo-LRU policies: Not Recently Used (NRU) and Binary Tree (BT). Without the help of true LRU's stack property, we propose a profiling logic that applies curve approximation methods to derive the hit curve (hit counts under varied way allocations) for an application. We then propose a hybrid partitioning mechanism, which mitigates the gap between the predicted hit curve and the actual statistics. Simulation results demonstrate that our proposal can improve throughput by 15.3% on average and outperforms the *stack-estimate* proposal by 12.6% on average. Similar results can be achieved in weighted speedup. For the cache configurations under study, it requires less than 0.5% storage overhead compared to the last-level cache. In addition, we also show that profiling mechanism with only one true LRU ATD achieves comparable performance and can further reduce the hardware cost by nearly two thirds compared with the hybrid mechanism.

key words: *cache partitioning, pseudo-LRU, curve approximation*

1. Introduction

Many cache partitioning techniques have been proposed to provide performance isolation for shared last-level cache (LLC) [1], [9], [13], [16]. Most proposals are based on true LRU policy and rely on its stack property to function. However, true LRU is typically not implemented in commercial processors, due to its hardware complexity and overhead. Instead, pseudo-LRU (PLRU) replacement policies, such as Not Recently Used (NRU) policy in SUN processors [19] and Binary Tree (BT) [8] in IBM processors, dominate. Therefore, those cache partitioning proposals based on the stack property of true LRU may not be beneficial in many commercial processor designs.

A cache partition algorithm can use the number of hits under different number of cache ways as its input to improve performance. Since the hit counts of an application can be modeled as a function of the number of cache ways allocated, we define a hit curve as the hit counts over a number of cache ways in this work. Auxiliary Tag Directories (ATD) can be used to derive the hit curve. Since PLRU policies don't have stack property, the hit curve cannot be directly gathered from only one n-way ATD (n is the associativity of cache) through one-pass execution [17]. Kedzierski et al. [6] proposed two partitioning mechanisms for PLRU policies NRU and BT respectively. We note them as *stack-estimate* mechanisms. In both mechanisms, an estimation scheme for hit curve is proposed requiring only one n-way ATD. In other words, all the hit counts under allocations from 1 to n ways can be estimated through one n-way ATD. The stack-based replacement policy posses the inclusive property. For example, a cache block existing in the 4-way cache must exist in an 8-way cache. However, PLRU policies that don't have stack property can't guarantee the above inclusive character. As a result, a hit in 4-way ATD may miss in 8-way ATD. Moreover, a hit in the 4th way of an 8-way ATD may also miss in the actual 4-way ATD. The stack-estimate mechanisms don't take the impact of different number of ways in ATDs into account, and actually are developed under the assumption that PLRU policies have stack property. Therefore, the estimation of hits and misses via the traditional method using only an n-way ATD may result in large deviation from the reality. The inaccuracy in hit curves probably degrades system performance.

One way to improve the accuracy of hit curve is to use n separated ATDs, from 1-way ATD to n-way ATD. Each ATD is accessed by the same access sequence. Thus, cache blocks are evicted or updated independently without the interference among applications. We note this method as fullset. However, fullset incurs too much hardware overhead. To reduce the overhead, we propose an estimation method requiring only a few ATDs to get sampling points on the hit curve, and then estimates the entire hit curve through curve approximation. To further improve the performance of profiling logic, we propose a hybrid partition mechanism that dynamically selects partitions and achieves most benefits.

The paper makes the following contributions:

Manuscript received January 3, 2013.
Manuscript revised April 27, 2013.

[†]The authors are with Key Laboratory of Trustworthy Distributed Computing and Service of Ministry of Education, Beijing University of Post and Telecommunications, Beijing, 100876 China.
[††]The authors are with Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing, 100084 China.
a) E-mail: zhangx@bupt.edu.cn

DOI: 10.1587/transinf.E96.D.2514

- A curve approximation method is proposed to estimate the hit curve for PLRU replacement policies, without the prerequisite of true LRU's stack property.
- Based on the curve approximation method, we implement a low-overhead profiling mechanism to derive the estimation hit curve.
- A hybrid mechanism is proposed to dynamically select the most optimal partitioning mechanisms to further improve performance, which can adapt to varying scenarios.
- Extensive simulations show our proposal can improve throughput by 15.3% on average under NRU and 11.0% on average under BT against non-partitioned baseline, and significantly outperforms the recently proposed PLRU partitioning mechanism stack-estimate by 12.6% on average. It requires less than 0.5% of storage compared with 4MB LLC.
- Through evaluation, we also observed that the profiling mechanism with only one true LRU ATD could provide comparable performance as the hybrid mechanism. Though not as adaptive as the hybrid one, it can further reduce hardware cost by nearly two-thirds.

2. Related Works

Partition with profiling logic: Cache utilization under varied cache sizes needs to be obtained by low-overhead profiling logic to determine the right partition. Zhou et al. [23] proposed one operating system (OS) implementation and one hardware monitor to dynamically track Miss-Ratio-Curve (MRC) of applications at run time. RapidMRC [20] is also an OS-based technique to obtain L2 cache MRC. Qureshi and Patt [13] proposed a low-overhead profiling circuit which uses a group of auxiliary tags having the same associativity as main cache and dynamic set dueling scheme (DSS) to track the utility of cache under varied number of ways, and this mechanism is referred to as TLRU-UCP in this work. Suh et al. [18], on the other hand, proposed an in-cache monitoring scheme to track the utility of cache without auxiliary tags. However, the interference among applications in the main cache degrades the the accuracy of profiling. All these proposals are based on the true LRU policy. Kim et al. [7] proposed a dynamic partitioning mechanism that is independent of replacement policies; however, it is only able to lock in local optimal partitions.

Partition with enforcement logic: Cache can be partitioned in ways [2], [13], [22], in sets [14], [16], [21] or in finer-grained blocks [15]. Set-partitioning schemes require disjoint address spaces across applications, the assumption of which does not hold in many cases. Way-partitioning schemes strictly enforce partitions within a fixed set of options and may restrict performance improvement. Therefore, some relaxed enforcement logic is desired. Some proposals [4], [5] partition cache approximately by alternating insertion and replacement policies, however, they cannot guarantee landing in the desired partitions. Promotion-insertion pseudo-partitioning (PIPP) [22] resolves the above problem and attains target partition at the cost of applying LRU policy, which is hard to apply to PLRU policies. Stack-estimate [6] introduces enforcement logics that fit for NRU and BT, but they strictly enforce the target partitions within a predetermined set of options, and can't adapt to frequent changes in access patterns.

3. Design

3.1 Motivation

The accurate hit curve of each application is crucial to compute the optimal partitioning. Figure 1 shows the hit curves of LRU, NRU and BT. The single-threaded applications shown are drawn from SPEC 2006, and L3 cache (last-level cache) contains 4096 sets with varied associativities[†]. We observe that in these cases, the hit curves of BT and NRU resemble those of LRU. This phenomenon motivates our proposed estimating method, that is, using hit curve of the conceptual LRU to estimate the cases of PLRU policies. While in other cases that hit curves of PLRU doesn't resemble those of LRU, we use linear approximation or other estimation methods according to specific conditions.

3.2 Framework

Figure 2 depicts the framework for supporting the PLRU partitioning on a quad-core system. Each core is assigned an ATD and two groups of hit counters tracking the hit curve of an application. The dedicated ATD for each core contributes to tracking the hit curves without the interference from other cores. The number of ATD sets is reduced as prior research on TLRU-UCP which suggests that 32 sets are enough to attain the track accuracy for the entire cache. Each access to the shared LLC is recorded by the corresponding ATD. In TLRU-UCP, one n-way ATD (n is the associativity of shared cache) is required to track the LRU hit curve that can obtain hit counts from 1-way to n-way. Due to the lack of stack property, the primitive implementation depends on n groups of ATDs (from 1-way to n-way) to track the hit curve of PLRU policies. This results in the overwhelming complexity and overhead.

In our mechanism, we reduce the hardware overhead by using a few sampling ATDs of 3-way, 6-way, 9-way and 13-way associativities[††], as is shown in the shaded area in Fig. 2. Our sampling ATDs apply PLRU instead of LRU. At the same time, ATD for tracking LRU hit curve with true LRU replacement policy is still reserved, as is shown with "set 16way" in Fig. 2. The true LRU is not implemented in LLC, and thus the overhead is minimal. Once the profiling phase is done, the LRU hit curve and the sampling points

[†]More information on simulation environment is available in Sect. 4.

[††]The principle to choose sampling points is explained in Sect. 3.3. Our proposed method generally applies to other cache configurations.

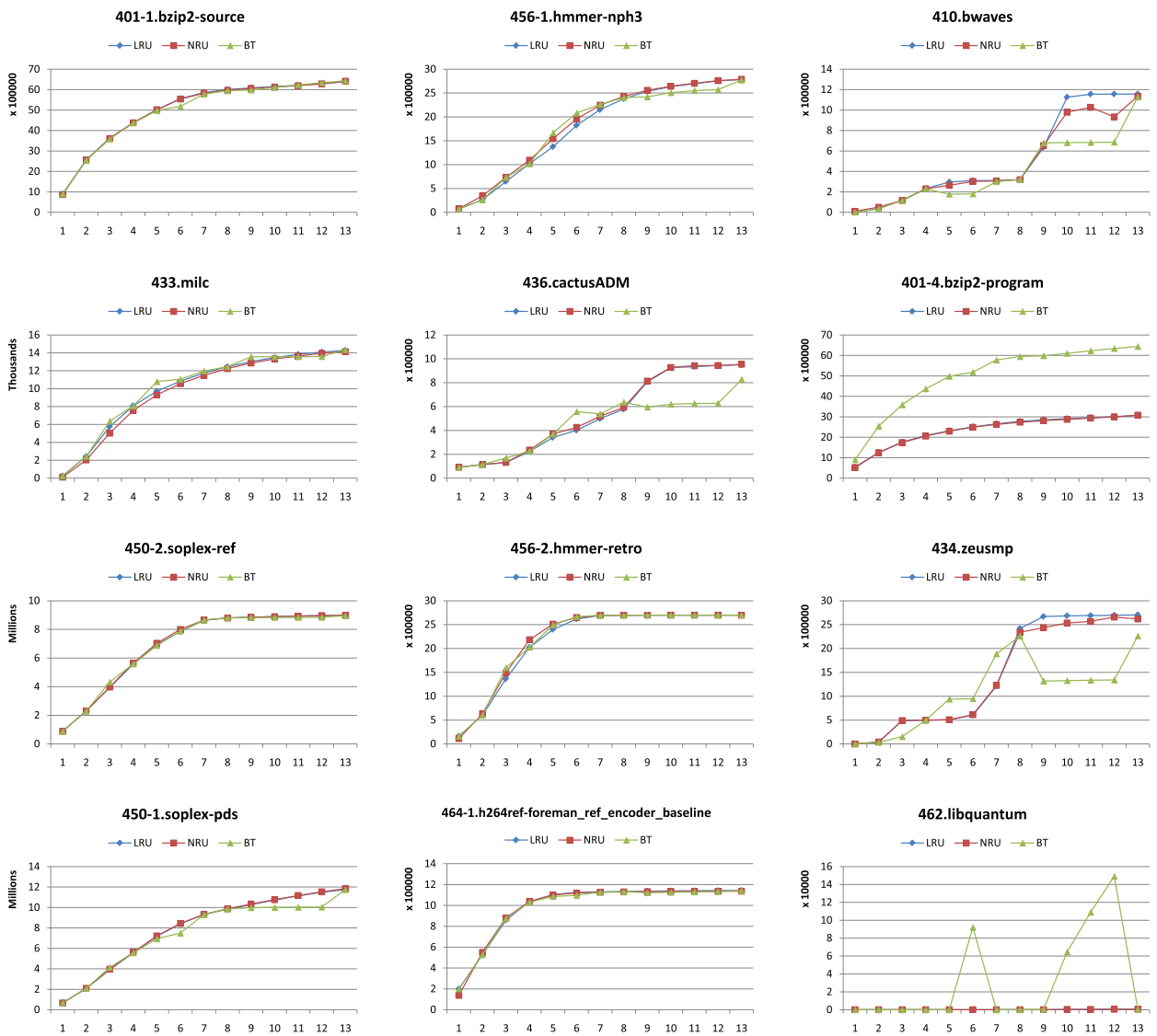


Fig. 1 Hit curves of LRU, NRU and BT. The x-axis denotes the number of ways allocated from 1-way to 13-way. The y-axis labels the corresponding hit counts.

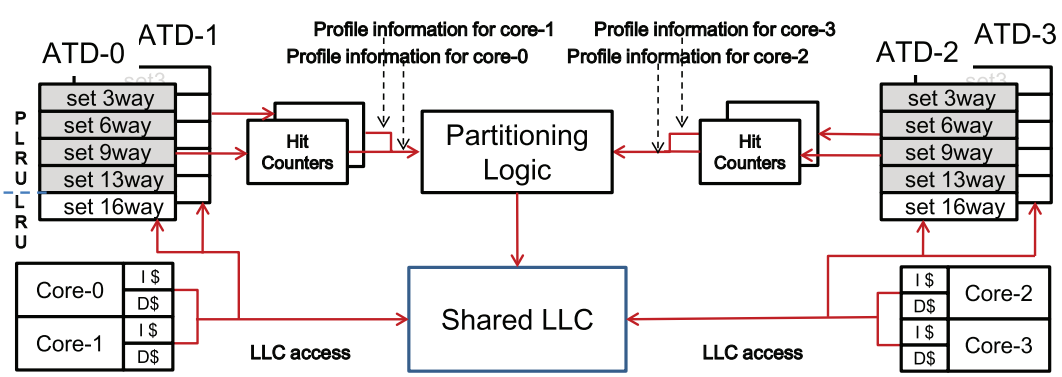


Fig. 2 Hardware support for Pseudo-LRU Cache Partitioning.

on PLRU hit curve are combined to estimate the unsampled points. As a result, the whole PLRU hit curve can be derived. Note that, two groups of hit counters are required: one is for storing the hit counts for LRU policy; the other is for PLRU policy.

3.3 Curve Approximating

The curve approximation method is proposed, which uses the hit curves of LRU to estimate the hit curves of PLRU. Merely one LRU hit curve may not be adequate when some PLRU hit counts don't match LRU hit curve. In this case, we use the curvilinear translations of LRU hit curve as a supplement.

By comparing the hit counts of the sampling points on PLRU hit curve and the corresponding neighboring points on LRU hit curve, we estimate the difference between the LRU hit curve and the PLRU hit curve. Based on this difference, hit counts of other unsampled points can be derived by interpolation. The number of sampling points should be appropriate, since too many sampling points will cost too much hardware, while too few sampling points may incur too much error. For example, in a 16-way cache shared by 4 threads, each thread occupies at least one way. Therefore, one thread can occupy at most 13 ways. In total 13 points are required to derive the hit curve. We choose four points as sampling points: 3, 6, 9 and 13. The hit count of point 1 under PLRU is always the same as that under LRU. The hit counts of the other 8 points are obtained through estimation. The sampling intervals are chosen with the incremental order, e.g. interval 1 between points (1, 3), interval 2 between points (3, 6) and (6, 9), and interval 3 between points (9, 13). It makes sense because the hit counts with smaller allocated ways, are more frequently used in partitioning algorithm, and thus need to be more accurate. In addition, the reason why we don't choose sampling points that are the power of two (e.g. 4, 8) will be explained in Sect. 3.4.

Once the sampling points are decided, the hit curves can be seen as piecewise functions among these sampling points. There are four pieces: 1, 3, 3, 6, 6, 9 and 9, 13. Other unsampled points can be derived using the interpolation, based on the value of sampling points and the LRU hit curve (or its translations). The parameters are defined as follows:

m, n	x-axis of left starting and right end of a piece
$H_{lru}(k)$	hit counts with k ways under LRU
$H_{plru}(k)$	hit counts with k ways under pseudo-LRU

Note that a piece can contain three, four or five points, and the associated approximation methods are quite similar. Due to space constraints, we take a piece with four points as an example, and describe the curve approximation method in the following.

(1) Initialization

For the starting point m, we first find the points m_a and m_b on the LRU hit curve that are just above and below point m. Then, we can determine if these two neighboring points are very close to point m.

First, find neighboring points on LRU. For each sampling point m, initialize m_a and m_b, where

$$H_{lru}(m_a) \geq H_{plru}(m) \geq H_{lru}(m_b)$$
$$m_a = m_b + 1;$$

Then, determine whether m is close to m_a and m_b on LRU. m_s is defined to indicate that close point where $m_s \in \{m_a, m_b, NULL\}$. Then m_s can be obtained:

$$\begin{aligned}
&if(\left|H_{lru}(m_a) - H_{plru}(m)\right| < (1/16) * H_{plru}(m))\\
&\quad m_s = m_a;\\
&else\ if(\left|H_{lru}(m_b) - H_{plru}(m)\right| < (1/16) * H_{plru}(m))\\
&\quad m_s = m_b;\\
&else\ \ m_s = NULL;
\end{aligned}$$

Boundary conditions: (1) if m has only one neighboring point above, m_b is set to $NULL$; (2) if m has only one neighboring point below, m_a is set to $NULL$.

The above initialization procedure is also fit for sampling point n when simply replacing m with n. The initialization of n_s can be traced by analogy.

(2) Estimate the non-sample points

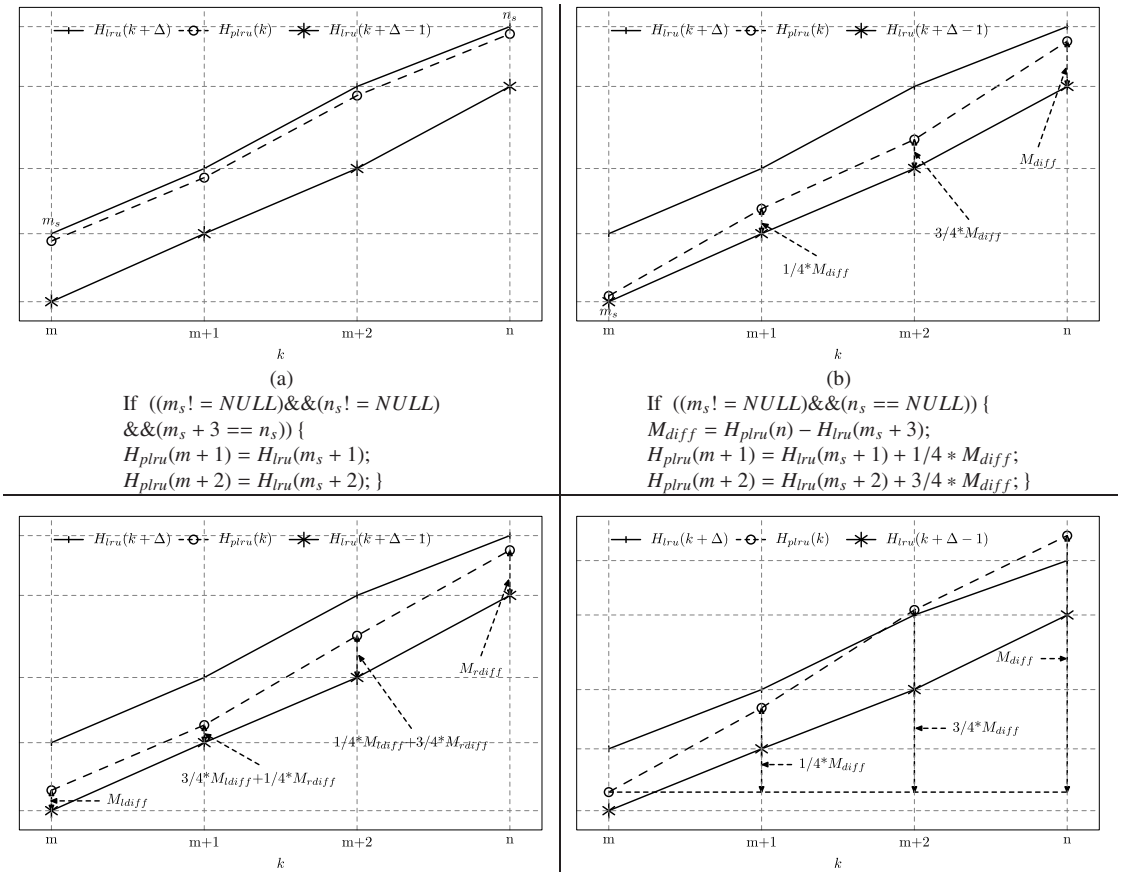
According to similarity between the PLRU hit curve and the neighboring LRU hit curve (or its curvilinear translations), the estimation methods are categorized into four cases (as shown by Fig. 3):

- if the two sampling points (m and n) are both approaching the same LRU hit curve (or its translation), we use the LRU hit curve (or the translation) to represent the PLRU hit curve (Fig. 3 (a)).
- If only one sampling point is approaching the LRU hit curve (or its translation), we use a triangle approximation. Figure 3 (b) shows the case in which the left starting point m is close to a point m_s on LRU hit curve. Similar method can be applied to the other case.
- If neither of the two sampling points are approaching the LRU hit curve (or its translation), but they are in the region between two LRU hit curves (or its translations), we use trapezoid approximation (Fig. 3 (c)).
- Otherwise, we use linear approximation (Fig. 3 (d)).

3.4 Hybrid Partitioning Mechanism

Figure 4 shows an example of the hit curves under different mechanisms. LRU denotes the LRU hit curve obtained from TLRU-UCP mechanism. Fullset mechanism uses ATDs ranging from 1-way to 13-way to derive PLRU hit curve. Sampleset is our proposed curve approximation mechanism. The data are obtained from the profiling logic in experiment.

Figure 1 and Fig. 4 show that the BT hit curve is not as smooth as that of LRU (or NRU). The variations make BT hit curve hard to approximate. The reason is that the binary tree is not symmetrical when the number of ways is not the power of two. As a result, the blocks within the subtree with fewer blocks generally have high probability to be evicted, which leads to the thrashing in its hit curve. Examples can be seen in Fig. 4 and 410.bwaves in Fig. 1: on point 4 and 8 that are 2^n, hit counts of BT are nearly the same as LRU, while for other points that are not 2^n, hit counts of



(a)

If $((m_s != NULL)\&\&(n_s != NULL)$
$\&\&(m_s + 3 == n_s))$ {
$H_{plru}(m+1) = H_{lru}(m_s + 1)$;
$H_{plru}(m+2) = H_{lru}(m_s + 2)$; }

(b)

If $((m_s != NULL)\&\&(n_s == NULL))$ {
$M_{diff} = H_{plru}(n) - H_{lru}(m_s + 3)$;
$H_{plru}(m+1) = H_{lru}(m_s + 1) + 1/4 * M_{diff}$;
$H_{plru}(m+2) = H_{lru}(m_s + 2) + 3/4 * M_{diff}$; }

(c)

If $((m_s == NULL)\&\&(n_s == NULL)$
$\&\&(m_a + 3 == n_a)\&\&(m_b + 3 == n_b))$ {
$M_{ldiff} = H_{plru}(m) - H_{lru}(m_b)$;
$M_{rdiff} = H_{plru}(n) - H_{lru}(n_b)$;
$H_{plru}(m+1) = H_{lru}(m_b + 1) + 3/4 * M_{ldiff}$
$+1/4 * M_{rdiff}$;
$H_{plru}(m+2) = H_{lru}(m_b + 2) + 1/4 * M_{ldiff}$
$+3/4 * M_{rdiff}$; }

(d)

Otherwise:
$M_{diff} = H_{plru}(n) - H_{plru}(m)$;
$H_{plru}(m+1) = H_{plru}(m) + 1/4 * M_{diff}$;
$H_{plru}(m+2) = H_{plru}(m) + 3/4 * M_{diff}$;

Fig. 3 Algorithms to compute the non-sampled points on PLRU hit curve. The algorithms are for pieces including 4 points, and thus $n = (m + 3)$. $\Delta = (m_a - m)$.

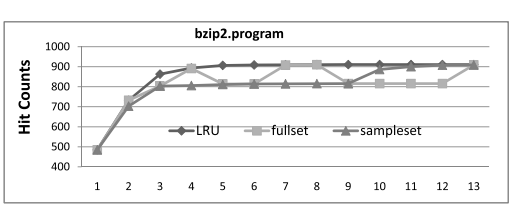


Fig. 4 An example to show LRU hit curve, and BT hit curves obtained from fullset and sampleset mechanisms. The x-axis denotes the number of ways allocated

BT are apparently different from that of LRU. Thus, LRU hit curve can be seen as an alternate approximation curve with sampling points 2^n, which is complementary with sampleset hit curve with sampling points that are not 2^n.

As a result, a hybrid partitioning mechanism denoted as hybrid-par is proposed as follows. Besides acting as the guide for curve approximation in sampleset, the LRU hit curve itself is also applied as the input to generate an alternate partitioning. The partitions coming from LRU hit curve and the PLRU hit curve are both candidates and competing for which is the better. The dynamic mechanism for choosing the better candidate partitioning is described in the following subsection. The hybrid mechanism potentially outperforms sampleset.

3.5 Enforcement Policy

After deriving the hit curve through the curve approximation method, partitioning algorithm is used to decide the optimal partitions. We used the partitioning algorithm proposed in [13] to get two candidate partitionings under sampleset and LRU respectively. Then a dynamic policy selection mechanism is implemented with the prior wisdom that uses Set Dueling technique [12] to identify which candidate partitioning is better suited.

To enforce target partitions obtained from hybrid-par, we adopt two enforcement policies for NRU: the *hybrid-strict* policy and the *hybrid-flexible* policy. *Hybrid-strict* policy strictly enforce the target partitions. *Hybrid-flexible* permits capacity stealing under target partitions to achieve further performance improvement. *Hybrid-flexible* is similar to the enforcement policy proposed in [13] for LRU. When implementing that policy under NRU, a Not-Recently-Used block takes place of an Least-Recently-Used block.

4. Evaluation

4.1 Simulation Setup

We use CMP$im [3], a Pin [10] based trace-driven x86 simulator, to evaluate our proposal. The parameters of baseline system are shown in Table 1. Non-partitioned shared LLC under the corresponding NRU or BT policy is used as the baseline. We use two metrics: IPC throughput and weighted speedup [11]. Let IPC_i denote the IPC of the ith application when it concurrently executes with other applications and $SingleIPC_i$ be the IPC of the same application when it executes in isolation. The metrics are given by:

$$Throughput = \Sigma(IPC_i) \quad (1)$$

$$Weighted\ Speedup = \Sigma(IPC_i/SingleIPC_i) \quad (2)$$

We choose sixteen applications from SPEC2006 with a variety of cache demand. We fast forward each workload for two billion cycles, and then collect traces for five hundred million cycles. Since not all the combinations of single-threaded workloads can get benefits from cache partitioning, we just selected 14 representative combinations (as shown in Table 2), in which TLRU-UCP scheme can improve the throughput compared against the non-partitioned baseline. We perform partitioning once every 0.5M total accesses to LLC.

4.2 Comparison of Partitioning Mechanisms

Figure 5 † shows the system performance for six partitioning mechanisms: fullset, sampleset, TLRU-UCP, stack-estimate, LRU-par and hybrid-par. LRU-par here refers to the mechanism whose profiling logic requires only one 16-way ATD under true LRU, and without PLRU ATDs. Since the true LRU policy generally has higher performance than PLRU policies, and the accurate true LRU hit curve is easy to get because of stack property, TLRU-UCP potentially has better performance than other schemes. In most cases, hybrid-par outperforms or has comparable results as its counterparts. On average, hybrid-par can achieve the throughput improvement by 13.6%, and nearly the same as TLRU-UCP. LRU-par can improve throughput by 13.2%.

†The results on arithmetic mean (avg.) are shown because for stack-estimate scheme, some results are negative. As a result, geometric mean (gmean) can't reflect the negative cases.

Table 1 Baseline system parameters.

Component	Parameter
Processing core	4-wide pipeline out-of-order
Instruction latency	1 cycle
L1 I-cache	32KB/4-way/LRU/1 cycle
L1 D-cache	32KB/8-way/LRU/1 cycle
L2 cache	256KB/8-way/LRU/10 cycles
L3 (LLC) cache	4MB/16-way/30 cycles
Memory access latency	200 cycles
Cache line	64B

Table 2 Multiprogrammed workloads.

Mix-1	astar.rivers,h264.ref_baseline,libquantum.ref,milc.su3imp
Mix-2	bzip2.source,astar.rivers,lbm,soplex.ref
Mix-3	bwaves,bzip2.source,soplex.pds,soplex.ref
Mix-4	astar.rivers,hmmer.retro,bwaves,soplex.ref
Mix-5	bzip2.source,bwaves,soplex.pds,hmmer.nph3
Mix-6	hmmer.nph3,astar.rivers,h264.ref_baseline,bwaves
Mix-7	hmmer.nph3,soplex.pds,h264ref.sss_main,lbm
Mix-8	bzip2.program,astar.rivers,cactusADM,lbm
Mix-9	bzip2.source,zeusmp,lbm,hmmer.nph3
Mix-10	bzip2.program,bzip2.source,lbm,bwaves
Mix-11	h264.ref_baseline,cactusADM.benchADM,milc.su3imp,lbm
Mix-12	bzip2.source,h264ref.sss_main,lbm,bwaves
Mix-13	bzip2.program,bzip2.source,soplex.pds-50,lbm
Mix-14	bzip2.program,astar.BigLakes2048,hmmer.retro,lbm

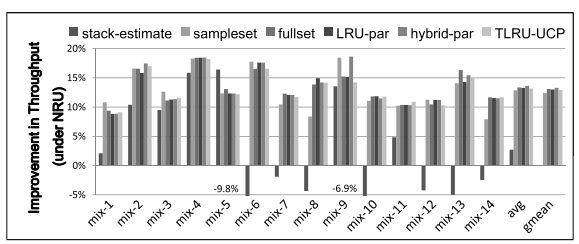


Fig. 5 Improvement in IPC throughput under NRU (against non-partitioned baseline).

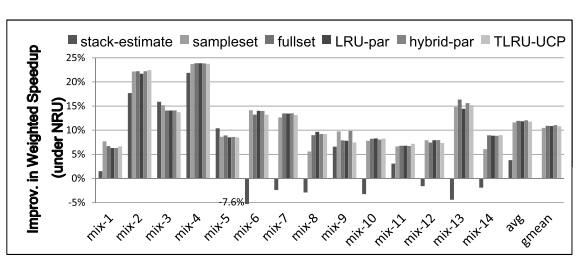


Fig. 6 Improvement in weighted speedup under NRU (against non-partitioned baseline).

Stack-estimate usually performs worse than other mechanisms, and in some applications (e.g. mix-6, mix-7), it degrades the performance. The average improvement in throughput under stack-estimate is 2.7%.

On average, the performance of LRU-par is only 3% worse than hybrid-par. The reason is that in many cases, the hit curves of PLRU resemble those of TLRU, and the performance gap between LRU-par and hybrid-par is rel-

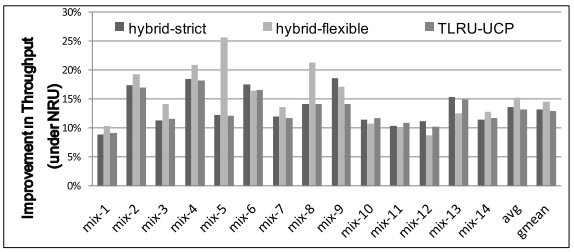


Fig. 7 Performance of enforcement policies (against Non-partitioned Baseline).

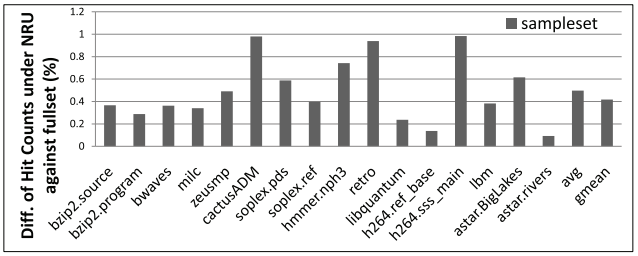


Fig. 8 Comparison of hit curves under NRU (against fullset).

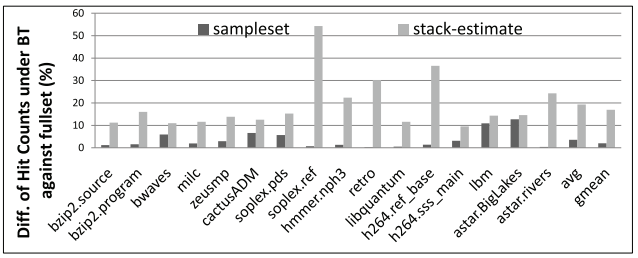


Fig. 9 Comparison of hit curves under BT (against fullset).

atively small. However, when referring to some specific cases, the performance gap gets larger. For example, for mix-9 in Fig. 5, the performance of LRU-par is worse than that of hybrid-par by 18.8%. In such cases, LRU-par may lead to worst-case execution time because of inaccurate hit curves. Things will get exacerbated in systems targeted for high-performance and hard-real-time use. Hybrid-par, on the contrary, is an adaptive partition mechanism to variable scenarios.

Figure 6 shows the weighted speedup compared with the non-partitioned baseline, and the performance trend is very similar to that in IPC throughput metric. The performance of fullset, LRU-par, hybrid-par and TLRU-UCP are close to each other. Sampleset increases the weighted speedup by 10.5%(on gmean), a little worse than fullset. On average, hybrid-par outperforms the other two schemes a little, and increase weighted speedup by 11.1% against the baseline, and the performance of LRU-par, fullset and TLRU-UCP are 10.8%, 10.9% and 10.8% respectively. The weighted speedup results demonstrate hybrid-par and LRU-par also provide better fairness. Due to the similarity of the trends between throughput and weighted speedup, we only deal with throughput in the rest of this paper.

The comparisons between fullset and sampleset show that fullset is not always superior over sampleset. The potential reason is that fullset is also an estimating method, which profiles a few sets to guide the entire partitioning, and uses the recent interval statistics to guide the partitioning in the next partitioning interval. Thus, it is a relatively trustworthy partitioning method. The performance of fullset partitioning is more stable than that of sampleset. It is shown that performance of fullset is comparable to that of TLRU-UCP, and both are better than that of sampleset. There is an anomalous case that stack-estimate performs best in mix-5. The reason is that the access pattern of mix-5 varies frequently. We will explain it in details in the next subsection.

4.3 Performance with Enforcement Policy

Figure 7 illustrates the impact of the enforcement logic with our hybrid-par mechanism. On average, hybrid-strict and hybrid-flexible promote the throughput by 13.6% and 15.3%, respectively. Compared with stack-estimate, hybrid-flexible improves throughput by 12.6%. It can be observed that our partitioning mechanism can be further improved with effective enforcement logic.

In mix-5, hybrid-flexible achieves the most significant improvement compared against hybrid-strict and TLRU-UCP. It can be inferred that the access pattern of mix-5 varies frequently. In this case, the methods such as sampleset and hybrid-par which use a long interval statistics to calculate the optimal partitioning, and strictly retain the partitions, limit performance. Hybrid-flexible can adapt to the varying access behaviors adaptively and thus provides better performance.

4.4 Comparison of Hit Curves

To analyze the accuracy of the curve approximation method and the reason why stack-estimate works not so well, we compare the hit curves of different schemes. Figure 8 compares the hit curves of sampleset compared with fullset under NRU. We obtain the hit counts under varied number of ways from different mechanisms after each interval, and normalize the difference to fullset hit counts. Results show that hit curve of sampleset resembles that of fullset by only a little difference of 0.49% on average (0.42% on gmean). It demonstrates that our proposed curve approximation method is accurate. Note that, result of stack-estimate is not shown in Fig. 8, because the profiling logic for NRU doesn't update hit counters when cache hit at a not-recently-used block in ATD. Thus, the hit counts of stak-estimate mechanism is much smaller than other mechanisms. Figure 9 shows the difference of sampleset and stack-estimate under BT against fullset. Sampleset provides a 3.6% difference on average (1.9% on gmean). Though the difference is bigger than that under NRU, it is reasonable considering the variations of BT hit curve. With stack-estimate scheme, the difference is 19.3% on average (16.9%

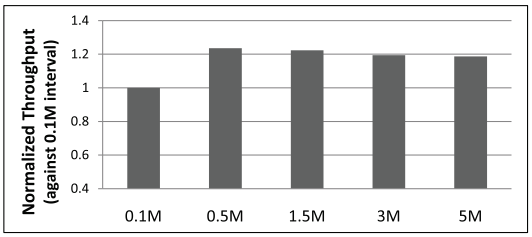


Fig. 10 Improvement in throughput under different partitioning intervals.

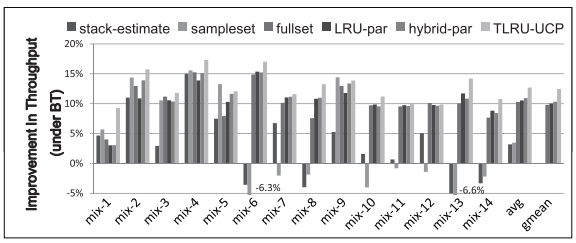


Fig. 11 Improvement in IPC throughput under BT (against non-partitioned baseline).

on gmean). It shows that our proposed curve approximation method can estimate the PLRU hit curve with good accuracy.

4.5 Impact of Partitioning Time Interval

Recall that we perform partitioning once every 0.5 million total accesses to LLC, and we also experiment with varied partitioning intervals including 0.1M, 1.5M, 3M and 5M accesses, results are shown in Fig. 10. It shows the performance of hybrid-par. Interval 0.1M has the worst performance, while for other intervals, there is no significant difference in performance. It is reasonable that applications generally have a long interval before facing a phase change. Perform partitioning in longer intervals mean we can can reduce the power overhead consumed in partitioning algorithms. In addition, we also experiment with varied number of sampling sets from 32 sets to 128 sets, results show that performance are quite similar.

4.6 Performance for BT

Figure 11 presents the performance of six partitioning mechanisms for BT against non-partitioned baseline. Fullset improves throughput by 10.3%, while the results of stack-estimate and sampleset are 3.1% and 3.5% respectively. Under BT, it is harder to estimate the accurate hit curve because the variation in hit curve of BT is larger than that of NRU, which exacerbates violation. When sampleset loses efficiency, the hybrid-par which can mitigate the variations is stable and can perform best or close to the best counterparts in most cases. Hybrid-par improves the throughput by 10.9%, only a little lower than TLRU-UCP, which improves the throughput by 12.7%. It is reasonable that BT fundamentally has lower performance, and BT hit curve is harder to be captured than LRU and NRU. The relative improvement between hybrid-par and stack-estimate is a noticeable 7.8%. LRU-par improves throughput by 10.5%, and it also outperforms stack-estimate and is a litter lower than that of hybrid-par. In weighted speedup metric, Fig. 12 shows that hybrid-par can provide 10.1% improvement, while the performance of LRU-par and TLRU-UCP are 9.7% and 11.4% respectively.

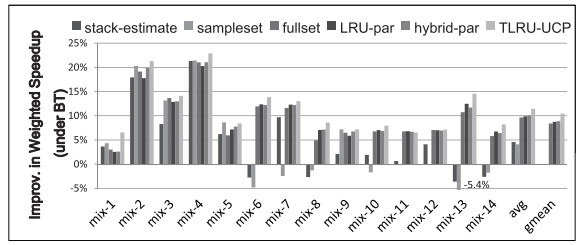


Fig. 12 Improvement in weighted speedup under BT (against non-partitioned baseline).

4.7 Overhead

The hardware overhead of our proposed hybrid-par is mainly from the profiling logic. Table 3 lists the breakdown of storage overhead of profiling logic for each core. The storage overhead is invariant to the number of cores, and doesn't increase with augment of set number, and thereby scalable. We assume 40-bit physical address space, 16-way shared LLC, and other parameters are based on our baseline system. Each profiling logic for NRU needs 4820B of storage (about 0.11% of the entire 4MB shared LLC), which indicates that about 0.44% storage overhead for implementing all four profiling logics. For other cache configurations, the similar sampling points are still applied to retain the storage overhead. Under BT policy, the storage overhead is almost the same as NRU because there is only one bit difference between BT and NRU in each sampling set. The storage overhead of fullest is 6196B, which is more than hybrid-par by 28.5%. For each access on the sampling set, hybrid-par only accesses 5 ATDs, while fullest accesses 13 ATDs. Therefore, the profiling logic of fullest consumes more area and power, while profiling logic of hybrid-par is relatively low-overhead and can be scaled to more cores. In addition to hybrid-par, under the same configuration, the storage overhead of the profiling logic of stack-estimate under NRU is 1600B for each core, while for LRU-par, that storage overhead is 1792B. Overall, compared to the 4MB LLC, the storage overheads of profiling logic for stack-estimate, LRU-par and hybrid-par under four cores are 0.14%, 0.16% and 0.44% respectively.

To evaluate the detailed hardware cost, we further implemented the TLRU ATD and NRU ATDs with Verilog-HDL and synthesized them with SYNOPSYS Design Compiler based on TSMC 0.18mm 1P6M CMOS technology. Results show that on 100MHz, the ATDs (with replace-

Table 3 Storage overhead of the profiling logic of hybrid-par for each application*.

	LRU Profiling logic(A_1)		**NRU Sample Profiling logic(A_2)**		**Total overhead(A_1 + A_2)**		
Each ATD entry	1b(v)+22b(tag)+4b(LRU)	27b	1b(v)+22b(tag)+1b(NRU)	24b	Profiling logic	1728+2976+64+52	4820B
# ATD entries/set		16	3+6+9+13	31	Baseline LLC	256KB tags+4MB data	4352KB
ATD overhead/set	27 bits/way * 16 ways	54B	24 bits/way * 31 ways	93B	**%increase vs. baseline**	4820B/4352KB	**0.11%**
Total ATD overhead	32 sampled sets*54B/set	1728B	32 sampled sets * 93B/set	2976B	TLRU-UCP overhead	1792B + (LRU bits - NRU bits) / 4 core	7936B
Hit counters	16 counters * 4B each	64B	13 counters*4B each	52B			
ATD + hit counters	1728B+64B	1792B	2976B+52B	3028B	**%red. vs. TLRU-UCP**	(7936B-4820B)/7936B	**39.3%**

*The calculations assume 40-bit physical address space, 16-way LLC cache with 64B block. No matter how many sets in LLC in total, 32 sampled sets are enough for sampling hit curve [13]. Thus, the total overhead of profiling overhead per application is invariant to number of cores and LLC capacity.

ment logic) in the profiling logic of stack-estimate, LRU-par and hybrid-par per core require 122K-gate, 123K-gate and 358K-gate respectively. Synthesized results also show that the area cost of ATDs in the profiling logic of stack-estimate, LRU-par and Hybrid-par per core is 0.45%, 0.45% and 1.3% respectively, compared to the 4MB 16-way 64B block L3 cache. The area of the L3 cache is estimated through the HP CACTI tool [24]. Overall, the hardware costs of all hree schemes are under a small threshold. Moreover, stack-estimate and LRU-par require less hardware and are easier to implement.

True LRU ATDs are required to implement in the LRU-par and hybrid-par scheme. The mechanism for True LRU used shift registers. For the 16-way ATD, it needs 4 bits for the way number, 1 valid bit by 16 entries. Each entry has a 4 XORs for matching ids. Thus, it requires (4+1)*16 master-slave latch, 4*16 XOR, 16 muxes plus some glue logic for each set. Since true LRU is not implemented in LLC, the overhead is minimal. One 32-set 16-way ATD under true LRU requires 123K-gate and the area overhead is only 0.45% of LLC area. In addition to true LRU mechanism, the curve approximation mechanism in sampleset and hybrid-par merely contains several add, compare and shift operations, which require simple circuit and negligible storage. One 10-bit policy selection counter is required to perform hybrid-par, together with the negligible logic for choosing sampling sets. The operation on ATD is not on critical path, and thus latency is not impacted. In addition, the estimation mechanism and partitioning algorithm are carried out once in every partitioning interval.

5. Conclusions

Previous studies have proposed a variety of mechanisms to ameliorate the performance of cache partitioning under LRU. However, most of these LRU-based techniques are based on the stack property, which is avoided by PLRU policies widely adopted by the commodity processors. Thus, practical cache partitioning mechanisms for PLRU policies are required. Previous PLRU cache partitioning methods don't take the impact of stack property into account, which leads to inaccurate hit curves and thus degrades system performance.

In this paper, we proposed the partitioning mechanisms specifically for PLRU policies. A curve approximation method is proposed to estimate the hit curve for PLRU policies. In addition, a hybrid mechanism (i.e. hybrid-par) is devised to dynamically select optimal partitioning for further performance improvement. Extensive simulation results show that our proposal can improve throughput by 15.3% under NRU (11.0% under BT), and achieves 12.6% (7.7% under BT) average improvement over recently proposed stack-estimate. We further verify our proposal in weighted speedup metric, and obtain the similar results. For the cache configurations under study, the storage overhead of hybrid-par is 0.44% and the area cost is about 5.2%, compared to the 4MB last-level cache. To further reduce the hardware overhead of hybrid-par, we observed that the profiling logic with only one true LRU ATD (i.e. LRU-par) also worked well for PLRU cache partitioning. Though this scheme is not as adaptive as the hybrid-par scheme, it is easier to implement in terms of hardware cost. This design only increases area by around 1.8% compared to the area of LLC. Hybrid-par and LRU-par posses different pros and cons, which increases the design space for CPU designers to explore.

Acknowledgements

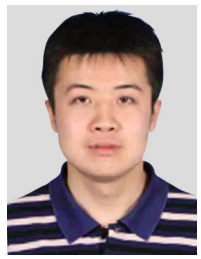
This work is supported by the Natural Science Foundation of China under Grant No.61300014, No.61202081, and also by the Fundamental Research Funds for the Central Universities [2013RC0301].

References

[1] J. Chang and G.S. Sohi, "Cooperative cache partitioning for chip multiprocessors," ICS '07, pp.242–252, 2007.
[2] D. Chiou, P. Jain, S. Devadas, and L. Rudolph, "Dynamic cache partitioning via columnization," DAC '00, 2000.
[3] A. Jaleel, R. Cohn, C. Luk, and B. Jacob, "CMP$im: A pin-based on-the-fly multi-core cache simulator," The Fourth Annual Workshop on Modeling, Benchmarking and Simulation (MoBS), co-located with ISCA 2008.
[4] A. Jaleel, W. Hasenplaugh, M. Qureshi, J. Sebot, S. Steely, Jr., and J. Emer, "Adaptive insertion policies for managing shared caches," PACT '08, pp.208–219, 2008.
[5] A. Jaleel, K.B. Theobald, S.C. Steely, Jr., and J. Emer, "High performance cache replacement using re-reference interval prediction (rrip)," ISCA '10, pp.60–71, 2010.
[6] K. Kedzierski, M. Moreto, F. Cazorla, and M. Valero, "Adapting cache partitioning algorithms to pseudo-lru replacement policies," IPDPS 2010, pp.1–12, April 2010.
[7] S. Kim, D. Chandra, and Y. Solihin, "Fair cache sharing and partitioning in a chip multiprocessor architecture," PACT '04, pp.111–122, 2004.

[8] H. Le, W. Starke, J. Fields, F. O'Connell, D. Nguyen, B. Ronchetti, W. Sauer, E. Schwarz, and M. Vaden, "IBM power6 microarchitecture," IBM J. Res. Dev., vol.51, no.6, 2007.
[9] 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," HPCA 2008, pp.367–378, Feb. 2008.
[10] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V.J. Reddi, and K. Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation," PLDI '05, pp.190–200, 2005.
[11] K. Luo, J. Gummaraju, and M. Franklin, "Balancing throughput and fairness in SMT processors," ISPASS '01, pp.164–171, 2001.
[12] M.K. Qureshi, A. Jaleel, Y.N. Patt, S.C. Steely, and J. Emer, "Adaptive insertion policies for high performance caching," ISCA '07, pp.381–391, 2007.
[13] M.K. Qureshi and Y.N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," MICRO 39, pp.423–432, 2006.
[14] P. Ranganathan, S. Adve, and N.P. Jouppi, "Reconfigurable caches and their application to media processing," ISCA '00, pp.214–224, 2000.
[15] D. Sanchez and C. Kozyrakis. "Vantage: scalable and efficient fine-grain cache partitioning," ISCA '11, pp.57–68, 2011.
[16] S. Srikantaiah, M. Kandemir, and M.J. Irwin, "Adaptive set pinning: managing shared caches in chip multiprocessors," ASPLOS '08, pp.135–144, 2008.
[17] R.A. Sugumar and S.G. Abraham, "Set-associative cache simulation using generalized binomial trees," ACM Trans. Comput. Syst., vol.13, pp.32–56, Feb. 1995.
[18] G.E. Suh, L. Rudolph, and S. Devadas, "Dynamic partitioning of shared cache memory," J. Supercomput., vol.28, pp.7–26, April 2004.
[19] Sun Microsystems Inc., "Ultrasparc T2 supplement to the ultrasparc architecture 2007," Draft D1.4.3., 2007.
[20] D.K. Tam, R. Azimi, L.B. Soares, and M. Stumm, "Rapidmrc: approximating l2 miss rate curves on commodity systems for online optimizations," ASPLOS '09, pp.121–132, 2009.
[21] K. Varadarajan, S.K. Nandy, V. Sharda, A. Bharadwaj, R. Iyer, S. Makineni, and D. Newell, "Molecular caches: A caching structure for dynamic creation of application-specific heterogeneous cache regions," MICRO 39, pp.433–442, 2006.
[22] Y. Xie and G.H. Loh, "PIPP: promotion/insertion pseudo-partitioning of multi-core shared caches," ISCA '09, pp.174–183, 2009.
[23] P. Zhou, V. Pandey, J. Sundaresan, A. Raghuraman, Y. Zhou, and S. Kumar, "Dynamic tracking of page miss ratio curve for memory management," ASPLOS '04, pp.177–188, 2004.
[24] CACTI. http://www.hpl.hp.com/research/cacti/

Xi Zhang received his B.S. degree from Harbin Institute of Technology, China in 2006, and his Ph.D. degree from Tsinghua University in 2012. He is now an assistant professor in School of Computer Science at Beijing University of Posts and Telecommunications (BUPT). He is also in Key Laboratory of Trustworthy Distributed Computing and Service of Ministry of Education. He has worked in the areas of multicore architecture, non-volatile memory (NVM) and trustworthy computing and services.



Chuanyi Liu received his PhD (2009) in computer science and technology from Tsinghua University, China. He is now an assistant professor of computer science & engineering at Beijing University of Posts and Telecommunications. He has broad research interests in computer systems, including architecture, file and storage systems, information security and data protection, he is now focus on cloud computing and cloud security. Dr. Liu spent one year as a visiting scholar at the Digital Technology Center of the University of Minnesota.

Zhenyu Liu received his B.E., M.E., and Ph.D degrees in electronics engineering from Beijing Institute of Technology, China, in 1996, 1999 and 2002, respectively. His doctoral research focused on the real time signal processing and relative ASIC design. From 2002 to 2004, he was a Post Doctorate Fellow at Tsinghua University of China, where his work mainly concentrated on the embedded CPU architecture design. From September 2004 to March 2009, he was the visiting researcher of the Graduate School of IPS in Waseda University. He is currently an Associate Professor of Tsinghua University, Beijing, China. His research interests include real-time H.264/AVC encoding algorithm optimization and associated VLSI architecture design.

Dongsheng Wang was born in 1966 in China. He received his B.E., M.E., and Ph.D degrees in computer science from Harbin Institute of Technology, China, in 1989, 1992 and 1995, respectively. He is now the professor of the Department of Computer Science at the Tsinghua University and the director of CPU and SoC Tech. Research Center, Tsinghua University. His research areas include computer architecture, Multicore and SoC, Disaster Recovery, and HA Computing.