NUcache: An Efficient Multicore Cache Organization Based on Next-Use Distance

R Manikantan† Kaushik Rajan‡ R Govindarajan†
† Indian Institute of Science, Bangalore, India
‡ Microsoft Research India, Bangalore, India

Abstract

The effectiveness of the last-level shared cache is crucial to the performance of a multi-core system. In this paper, we observe and make use of the DelinquentPC—Next-Use characteristic to improve shared cache performance. We propose a new PC-centric cache organization, NUcache, for the shared last level cache of multi-cores. NUcache logically partitions the associative ways of a cache set into MainWays and DeliWays. While all lines have access to the MainWays, only lines brought in by a subset of delinquent PCs, selected by a PC selection mechanism, are allowed to enter the DeliWays. The PC selection mechanism is an intelligent cost-benefit analysis based algorithm that utilizes Next-Use information to select the set of PCs that can maximize the hits experienced in DeliWays.

Performance evaluation reveals that NUcache improves the performance over a baseline design by 9.6%, 30% and 33% respectively for dual, quad and eight core workloads comprised of SPEC benchmarks. We also show that NUcache is more effective than other well-known cache-partitioning algorithms.

1. Introduction

Most multicores today have one or two levels of dedicated private caches, backed up with a shared last level cache (LLC). The performance of a multicore is heavily dependent on the performance of the LLC [19]. An efficient LLC can help reduce off-chip memory traffic and contention for memory bandwidth. The focus of this paper is on improving the design of the last level shared cache to improve overall system performance.

Traditionally caches have relied upon temporal locality among the accesses to decide on which blocks to retain longer. But the locality seen at LLC is often obscured by (i) interleaving of requests from multiple cores [16, 23, 8] and (ii) the filtering effect of caches closer to the processor [14, 17, 1, 18, 22, 15].

Several schemes have been proposed recently [8, 16, 23, 9] to manage the LLC in an efficient and fair manner. Of these proposals, [8, 16, 23] attempt to improve the performance of LLC by adapting some of the parameters of the cache, typically the number of cache ways allocated to each core [16] or by changing the insertion and promotion policies [23, 8]. In this paper we propose an alternative approach to mange the LLC that is tailor-made to utilize key characteristics of PCs that bring in a majority of lines into the LLC.

We observe that the notion of delinquent PCs [12, 21, 24], which has been commonly observed in private caches of single-core processors, holds (not surprisingly) in multicores as well. As delinquent PCs suffer a majority of misses, and lines are brought into the cache on misses (with the exception of prefetched lines), delinquent PCs are responsible for bringing in a majority of cache lines. The presence of only a few delinquent PCs indicates that there is an order of magnitude difference between number of addresses accessed and the number of PCs that are responsible for bringing lines into the cache. In this paper we test out the hypothesis that designing a LLC by prioritizing lines brought in by certain PCs over other lines can lead to better performance than a purely memory address reuse driven organization. To prioritize lines brought in by delinquent PCs over others, we propose our NUcache organization that can support higher associativity for lines brought in by some select PCs and lower associativity for the other lines. NUcache is a logical partitioning of cache associativity into MainWays and DeliWays. While all the incoming blocks are placed initially in the MainWays, only blocks brought in by a select subset of delinquent PCs, that are expected to bring additional hits, are placed into the DeliWays on replacement from the MainWays.

We find that simple heuristics like restricting the set of selected PCs to the top N delinquent PCs or to top PCs that contributed to a fixed percent of misses do not work well in practise. We analyze the behaviour of delinquent PCs in terms of the number of misses between the eviction of a line brought in by a delinquent PC and its subsequent access. We refer to this distance metric as the Next-Use distance[1]. We find that different PCs exhibit completely different next use behaviour. Hence a Next-Use agnostic solution that treats all PCs equally is unlikely to work well. Therefore one needs an intelligent way to select PCs to be able to make best use of the additional associativity.

We propose a cost benefit analysis based PC selection algorithm that uses the next use behaviour of delinquent PCs to choose a set of PCs that can make the best use of the DeliWays. The correlation between PC and Next-Use distance is the key factor that facilitates the proposed solution and acts as a link between the various components of the solution – NUcache and the PC selection mechanism. While Next-Use distance directly gives an estimate of the effort involved to convert a miss into a hit, the observation that only a few delinquent PCs exist, makes it feasible to track Next-Use distance in hardware.

We evaluate the proposed solution, NUcache together with the PC-selection mechanism, in the context of last level shared caches in multi-cores running multi-programmed workloads. We observed that NUcache improved the performance over a baseline LRU managed organization in the case of dual, quad and eight cores and for varying cache configurations. In the case of dual-core configurations, NUcache organization provides an average speed-up of 9.6% over a baseline LRU last level cache in terms of Average Normalized Turnaround Time (ANTT) metric [5]. For Quad and eight core workloads, the observed speedup over baseline LRU is 30% and 33% respectively. We compare NUcache with three recently proposed cache partitioning schemes, Utility based cache partitioning(UCP) [16], TADIP [8] and PIPP [23]. While these schemes are known to perform better than other contemporary schemes, NUcache performs better than all these schemes.

2. Motivation

2.1. Delinquent loads and Next-Use characteristic

Next-Use distance is associated with a cache block, and is defined as the *number of intervening misses to the associated cache set between the block's eviction to it being referenced again*. As cache replacements are required only on misses, Next-Use distance is a measure of how long should a block be retained, after it becomes the replacement candidate, to convert a miss in the baseline scenario into a hit.

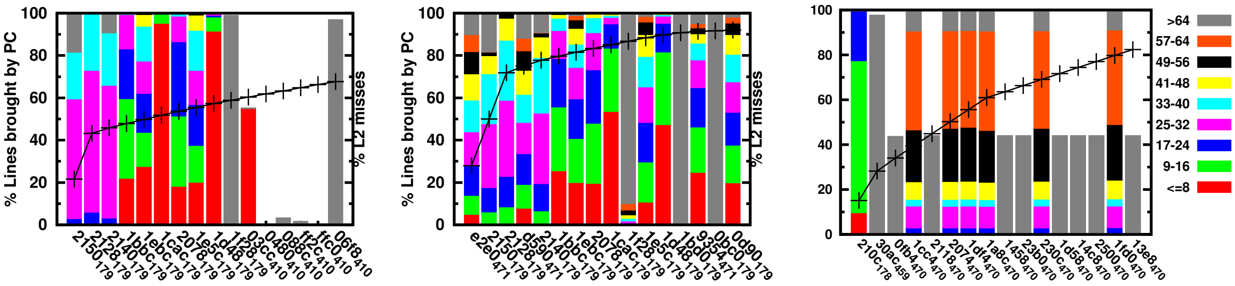
The term *Delinquent PCs* refers to a small set of static PCs that account for a significant fraction of misses in a program [12, 21, 24]. The presence of delinquent PCs has been used to target a variety of optimizations in single core processors [21]. We observe that delinquent PCs exist in shared LLC of multi-cores running multi-programmed workloads. For a variety of Dual, Quad and Eight core workloads, we observed that 16 delinquent PCs account for 50–95% of misses suffered at LLC.

Figure 1 shows the Next-Use distance of the top 16 delinquent PCs, for 2 dual-core workloads and one quad-core workload. Each graph is for a single workload which is composed of either two or four benchmarks. The line-graph in the Figure shows the cumulative fraction of L2 misses caused by the PCs under consideration. Each bar in the graph is for a single PC, uniquely identified using its 16 least significant bits (4 hex digits) and the benchmark from which it comes (shown as a subscript). The delinquent PCs in the graphs are displayed in descending order of misses caused by them. For each PC, we show the fraction of lines brought in by that PC that fall under various Next-Use distance ranges (in steps of 8 up-to 64 and $>$ 64 as another range). For instance, in *(410.bwaves,179.art)*, the most delinquent PC belongs to *179.art* and it accounts for around 20% of the misses caused at L2. 60% of the lines brought in by this most delinquent PC have a Next-Use distance between 25-32, i.e., upon eviction from the cache, they are referenced again within the next 25-32 misses. Another 20% of lines brought in by this PC have a Next-Use distance between 33-40. It is important to note that the subsequent access to an evicted block need not necessarily come from the same delinquent PC which brought it into the cache initially. Also some of the blocks brought in are never referred again after being evicted from the cache. This is reflected in the fact that some of the bars corresponding to individual PCs do not reach 100%.

2.2. Need for intelligent PC selection

As stated before, in this paper we test out the hypothesis that designing a LLC by providing lines brought in by certain selected PCs exclusive access to additional ways (called DeliWays) can lead to better performance than a purely memory address reuse driven organization that treats all lines equally. We argue that just looking at the fraction of lines brought in by a delinquent PC does not give sufficient information to choose one PC over another. One has to consider the correlation between the delinquent PC and its next-use distance characteristics. For example, from workload *(471.omnetpp, 179.art)* consider the top 2 delinquent PCs, $PC_1 = e2e0_{471}$ and $PC_2 = 2150_{179}$. Consider a cache in which 8 DeliWays are used to retain lines brought in by one of theses PCs for longer. From the graph it can be seen that PC_1 contributes to about 30% misses at the LLC and PC_2 contributes to 20% of the misses. Therefore on an average, 3 out of every 10 lines brought into the LLC is by PC_1 and one out of every 5 is brought in by PC_2. If PC_1 was the only selected PC, then, approximately once every 3

[1] Earlier work also refers to it as Eviction-Use interval [1].



(a) (410.bwaves,179.art)

(b) (471.omnetpp,179.art)

(c) (178.galgel,459.GemsFDTD,168.wupwise,470.lbm)

Figure 1. Delinquent PC – Next-Use correlation in Multi-Programmed Workloads

misses a line would enter the DeliWays. Assuming a FIFO replacement in the DeliWays, the line can stay there until a further 8 lines get pushed into the DeliWays. Considering the rate at which PC_1 brings lines into the cache, a line can stay in the DeliWays for 24 misses on average after the line entered DeliWays. Similarly, if PC_2 was the only selected PC, then once every 5 misses a line would enter the DeliWays and the line can stay in the DeliWays for 40 misses after the line entered DeliWays.

Now to estimate how many of the lines pushed into DeliWays will experience a hit, one has to know how long after eviction from the MainWays will a subsequent access to the line be seen. This is precisely the information that the Next-Use histogram provides. For PC_1, as long as the next-use of the line pushed into the DeliWays is within the next 24 misses it will see a hit. From the figure it can be seen that 25% of lines brought in by PC_1 have accesses within a next use distance of 24 and hence providing $25 \times 0.3 = 7.5\%$ reduction in misses if PC_1 uses the DeliWays. On the other hand about 70% of lines brought in by PC_2 have a next use distance within 40. This will lead to a $70 \times 0.20 = 14\%$ reduction in number of misses at the LLC. It can be seen that intelligent selection of PCs in this case can double the potential benefits.

Our PC selection mechanism is motivated by the need to perform this kind of cost-benefit analysis. It uses Next-Use histograms to estimate benefits to be gained by allowing a PC to access the DeliWays while accounting for the cost of pushing more lines into the DeliWays. Note that in addition to considering individual PCs our proposed algorithm considers combinations of PCs as well. In the above example it will also consider selecting both PC_1 and PC_2 for accessing the DeliWays. The complete algorithm is described in the next section.

In an equivalent LRU based baseline cache, all the lines will have access to the additional 8 DeliWays (as MainWays) per set. Any line that enters the last 8 ways, should see its Next-Use before the next 8 misses take place to experience a hit. Otherwise the line is evicted from the cache. It is commonly observed that LRU does not work well at the last level cache [16, 8, 23]. We empirically find that our proposed PC selection mechanism consistently outperforms LRU.

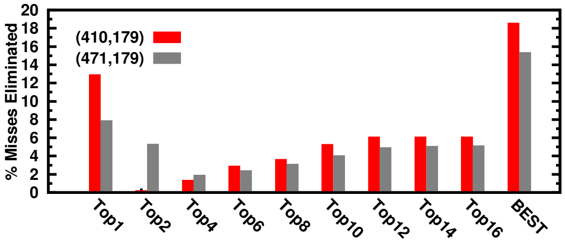


Figure 2. Naively selecting top-N delinquent PCs Vs Proposed scheme (BEST) – % of Misses likely to be eliminated when 8 Ways are used to retain lines longer for (410,179) and (471,179)

2.3. Other observations

From Figure 1 it can be observed that the Next-Use characteristic for different delinquent PCs is significantly different. This implies that a solution that is agnostic of Next-Use and simply selects a fixed number of most delinquent PCs to access DeliWays is unlikely to work. This can be observed from Figure 2 where we plot the % of misses eliminated by a naive PC selection algorithm that picks the top 8 or 16 delinquent PCs.

Secondly, the same PC can exhibit different Next-Use characteristic in different workloads. This can be observed by comparing the delinquent PCs belonging to 179.art in the workloads (410.bwaves, 179.art) and (471.bwaves, 179.art). This motivates the need for dynamically learning the Next-Use characteristic of delinquent PCs. In our proposed solution we introduce auxiliary structures to help track delinquent PCs and their next-use distance histogram.

2.4. Comparing other distance metrics

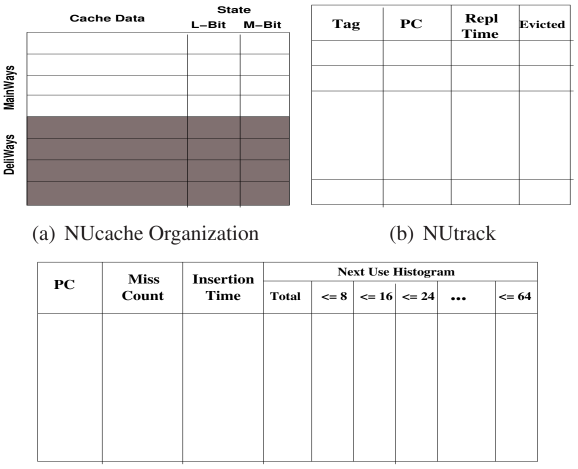
Reuse distance [6] and time distance [6] are other distance metrics that have been used to summarize the performance of caches. Time distance is measured as the number of intervening accesses between two successive accesses to a cache block and reuse distance is the number of *unique* accesses between two successive accesses to a cache block. In any cache, new lines are brought in and existing lines are replaced only on misses. Next-Use distance by virtue of measuring distance in terms of misses, provides a measure of the effort required to retain a block until its next use. For instance, a Next-Use distance of 10 means that once a block is evicted, it will be brought back in again after 10 misses. The effort required to avoid this subsequent miss, based on the information provided by Next-Use distance, is that the line should not be a replaced during the subsequent 10 misses once it becomes the candidate for eviction.

Unlike Next-Use distance, time distance cannot provide an accurate estimate of how far away the next access to the block is in terms of cache replacement decision. This is because time distance is measured in terms of intervening accesses which could be either cache hits or misses. For example, consider two blocks A and B with time distance of 100 (90 hits and 10 misses) and 50 (40 hits and 10 misses) respectively. While time distance says it takes twice as much effort to retain A until its next access compared to B, in truth the effort involved to retain A or B until their next access is the same. They should not be replaced during the next 10 misses. Even reuse distance suffers from the problem of hits and misses being treated at par. Also the uniqueness criteria makes it hard to measure reuse distance at runtime. In short, we claim that it is not possible to directly use reuse/time distance in place of Next-Use distance in our proposed solution without compromising its simplicity.

3. The Next-Use cache (NUcache)

3.1. Cache organization

NUcache is a logical partitioning of a cache's associativity into MainWays and DeliWays. Figure 3 shows the organization of one 8-way set in an NUcache with 4 DeliWays. In the figure, DeliWays are the shaded portion in the set, and are used *on demand* to store lines brought in by a select set of PCs longer. The organization shown in the figure can retain a maximum of 4 lines longer. The two extra *State* bits are used as follows: (i) the M-bit when set indicates that the line belongs to the MainWays, and (ii) the L-bit indicates whether a line belonging to the MainWays can enter the DeliWays in future. We use LRU replacement policy for the lines belonging to the MainWays. For the DeliWays, a simple FIFO replacement policy is better suited than LRU as once a hit is seen, the line has already provided the desired extra benefit. The setting of the M and L bits and the replacement mechanism are described below.



(a) NUcache Organization

(b) NUtrack

(c) DeliTrack

Figure 3. NUcache organization and associated structures.

An access to the cache searches all the ways (MainWays and DeliWays) in the appropriate set. On a hit, apart from updating replacement information no other changes are made. On a miss, a replacement candidate is chosen as follows. The least recently used line belonging to the MainWays is first considered as the replacement candidate. If its L-bit is set, indicating that the line should be retained longer, the line is marked as part of DeliWays by resetting the M-bit. If the maximum associativity allocated for DeliWays has not been reached, the next least recently used line from the MainWays is considered as the replacement candidate and the above procedure is repeated. If in the process the maximum DeliWays associativity is reached, then the oldest entry in the DeliWays is chosen for replacement. Note that the DeliWays are utilized to retain lines longer only on a demand driven basis. This permits all the ways of a cache to be used by the MainWays if no PC is allowed access to DeliWays. The newly entering line replaces the victim and sets the M-bit. Further, if the incoming line is brought in by a selected delinquent PC, then the L-bit is set as well.

3.2. DeliTrack and NUtrack

Two auxiliary structures are added to track all PC related information required by the PC selection algorithm. These are shown in Figure 3, and are placed alongside NUcache.

DeliTrack: DeliTrack is used to identify delinquent PCs and store their Next-Use histograms. The structure is indexed using PC. We assume that PC information is available at LLC as in earlier work on delinquent PCs [21, 13, 10](The recently held cache replacement contest (in conjunction with ISCA 2010) also assured/assumed the availability of PC at LLC.). An entry of DeliTrack is assigned for each PC that is currently being tracked. On a primary miss, if the PC is already being tracked, its miss count is incremented by one. Otherwise the least recently used entry in the DeliTrack is replaced with the PC suffering the miss. The usage of LRU in DeliTrack naturally helps retain the most delinquent PCs, while throwing away PCs that rarely suffer a miss. Upon insertion, the miss count is set to one and the insertion time stamp is updated to reflect the logical insertion time. The number of misses seen at a particular cache level is used as logical time. At any instance, the current logical time, the insertion time stamp and the miss count can be used to calculate the fraction of misses caused by a particular PC. This information is used in our PC selection algorithm described later in the section.

DeliTrack also stores the next use histogram for delinquent PCs. When a new PC is inserted into the DeliTrack, the corresponding histogram is cleared and then entries are subsequently populated through the NUtrack as described later. The histogram can store Next-Use distances from 8 to 64 in steps of 8.

NUtrack. NUtrack is a set-associative structure used to measure the Next-Use distance of lines brought in by delinquent PCs. NUtrack is indexed using block address. The key operations associated with NUtrack are insertion, replacement and learning. **Insertion:** When a delinquent PC experiences a miss and brings in a new line, an attempt is made to insert the line in NUtrack. If a free entry is present, insertion can take place. Otherwise, if the oldest entry (with evicted bit being set) has stayed beyond the maximum Next-Use distance tracked, 64 in this case, it is replaced and the newly arriving line is inserted. Insertion, sets the PC information and clears the evicted bit and replacement timestamp. If none of the above conditions hold, insertion does not take place. **Learning:** When a block is evicted from the cache, NUtrack is looked up to see whether it was inserted earlier (when it was brought into the cache originally). If so, the evicted bit of the NUtrack entry is set to indicate the beginning of the learning phase for the block. Setting the evicted bit during replacement time allows us to measure Next-Use distance from the time of replacement. Also the replacement time stamp is set to the value of global miss count. **Replacement**: Upon any cache miss to a block, if the block is present in NUtrack, it indicates a successful identification of Next-Use for that block. The Next-Use distance for the block is computed as the difference between the current value of global miss count and replacement time stamp. To arrive at a per-set level value for Next-Use, the distance computed above is scaled down by the number of sets in the cache. The PC field of the replaced NUtrack entry is used to update the appropriate fields of DeliTrack and the NUtrack entry is freed.

Note that even-though the Next-Use distance is measured only from the time of eviction to a subsequent access, a NUtrack entry is created as soon as the line enters the last level cache. This is done to avoid having to add an additional field that stores PC information along with the cache blocks.

```
selectPCs (DeliTrack, DeliAssoc){
  /*The structure S holds information
     related to selected PCs*/
  S.PCs=∅; S.NUhist={0}; S.MissFraction =0.0;
  while(true){
    BestGain = 0; BestCandidate = NULL;
    foreach(Candidate.PC P ∈ DeliTrack){
      if(P ∈ S.PCs) continue;
      CurrentGain = computeHits(S, P, DeliTrack
          , DeliAssoc);
      if(CurrentGain > BestGain){
          BestGain = CurrentGain;
          BestCandidate = P;
      }
    }
    if(BestCandidate != NULL){
      /*A new PC was identified*/
      S.PCs = S.PCs ∪ BestCandidate;
      S.MissFraction += BestCandidate.
          missFraction;
      S.NUhist += BestCandidate.NUhist;
    }
    else
      /*No New PC can increase hits*/
      return S.PCs;
  }
}
```

Figure 4. PC selection algorithm

3.3. PC selection mechanism

In this section, we discuss a cost-benefit algorithm to determine the PCs that should have access to the DeliWays. The goal of the cost-benefit algorithm is to maximize the number of hits provided by the NUcache organization. Once the PCs are identified, using the mechanism described below, lines brought in by them can be marked as selected for DeliWays by setting their L-bits to 1.

As it is impractical to evaluate all possible combination of PCs, we propose a greedy strategy that works incrementally by picking the delinquent PC that provides the maximum returns at each step. The algorithm is shown in Figure 4 and it operates on the contents of the DeliTrack. On each iteration it considers an as yet unselected *candidate* and computes the overall gains if it is added to the set of already selected PCs. This process is repeated until there comes a stage where no *candidate* provides a gain.

The estimation of benefits that a new *candidate* can provide is computed using the algorithm shown in Figure 5. The estimation of benefits is carried out by first computing the effective associativity of the DeliWays given the rate at which selected set of PCs push lines into it. This computation requires an addition and a left shift operation. The po-

```
computeHits($S$, $P$, DeliTrack, DeliAssoc){
  rate  = P.MissFraction + S.MissFraction;
  effectiveNextUseDist = Incr = rate << 3;
  gain = 0;
  for(HistIndex $\in${8,16,24,32,40,48,56,64}){
    /* The comparison below is effectively
         $HistIndex \leq (DeliAssoc/rate)$,
    where $(DeliAssoc/rate)$ is the average time a
         line can stay in DeliWays */
    if($effectiveNextUseDist \leq DeliAssoc$){
      gain += P.NUhist[HistIndex];
      gain += S.NUhist[HistIndex];
    }
    else
      break;
    effectiveNextUseDist += Incr;
  }
  return gain;
}
```

Figure 5. Algorithm to compute the gains when a candidate is added to the already selected set of PCs.

tential benefit is then calculated by going through the Next-Use histogram of the selected set of PCs and the *candidate* under consideration. We use one entry of the DeliTrack to store the Next-Use histogram of the set of PCs selected so far ($S.NUhist$). This allows us to avoid recomputing this information at each time step. Once the PCs are selected, we use a 16-entry structure (not shown in the Figures) to store them. We found that a 16-entry structure was enough in all the cases to store the set of selected PCs with access to the DeliWays. The PC selection algorithm is run once every 10,000 misses at the chosen cache level, L2 in our case. In our studies, for most of the workloads, this translated into a time interval of at the least 5 million processor cycles between successive runs of the PC selection algorithm.

With NUcache, the PCs picked in a multi-programmed workload scenario can come from any of the programs that constitute the workload. As the access to the DeliWays and the usage of the associativity present in the DeliWays are dictated by the selected PCs, the PC-selection mechanism implicitly leads to a partitioning of the DeliWays across the various programs. Also note that, while we use the criteria of maximizing hits to select the best PC at each step, it is possible to use a range of selection criteria to suit various design goals ranging from throughput to fair speedup of all the programs. The study of alternative selection criteria is left to future work.

3.4. Hardware requirements and PC selection overhead

The auxiliary structures introduced for the NUcache organization, DeliTrack and NUtrack are situated off the critical path. While DeliTrack is a predictor like structure indexed by PC, NUtrack requires a cache like organization. The space overhead for the structures is shown in Table 1. We assume a 40 bit physical address space and 32 bit counters for the appropriate fields of DeliTrack and NUtrack. The table also reports the storage overheads for the M and L bits in the physical cache organization, the global miss count register and the selected PCs table. For a 1MB cache, the storage overhead is only an additional 1.75% over a traditional LLC of the same size. Later in Section 4, we show that these structures (at the same size) perform well for a 4MB/8MB cache as well. Hence the storage overheads will be lesser with larger caches. This is primarily because of the fact that delinquent PCs, by definition, are expected to be few in number. Also as none of the structures track any information on a per-core basis, the proposed solution does not face hardware limitations in scaling to larger number of cores. We demonstrate this by using the same DeliTrack and NUtrack configurations for dual, quad and eight core configurations.

Next we describe the overhead in running the PC selection algorithm. The average number of PCs selected for the various workloads ranged from 1.5 to 8.6 with a median of 3. The maximum number of PCs selected across all the workloads is 12. Also not all 64 entries in the DeliTrack are delinquent PCs. A good fraction of them are recently inserted entries which lack adequate history to be considered by the algorithm. The algorithm used to compute the benefits provided by a set of PCs requires a maximum of 34 operations (25 additions, 1 shift operation and 8 comparisons). The average number of operations per single run of the PC-selection algorithm, ranges from 269.3 operations for the workload (471,179) to 6491.9 operations for the workload (410,179). Considering that the operations are simple and incur low latency, the PC selection algorithm finishes easily within a few thousands of cycles. As the algorithm is called only once every 5 million cycles or more, the overhead is less than 0.05%.

4. Experimental setup and performance metrics

4.1. Simulation methodology

We use M5 simulator [2] in system call emulation mode to carry out all the simulations. The benchmarks considered are drawn from SPEC2000 and SPEC2006 benchmark suites and are compiled for ALPHA ISA. We simulate dual, quad and eight core processors. All the processors have private L1 caches with a shared L2 cache. The machine parameters are presented in Table 2. We used several multi-programmed workloads. The workloads are discussed in detail later in the section. All the benchmarks in the multi-

DeliTrack		NUtrack	
PC Insertion timestamp Miss count Histogram (8 categories, 2 Bytes each)	5 Byte 4 Byte 4 Byte 16 Byte	Tag Insertion time PC	34 bits 32 bits 40 bits
Size per Entry Total Size (64 Entry)	29 Byte 1856 Byte	Size per entry Total Size (1024 entry)	106 bits 13568 Byte
Global Miss Count			4 Byte
List of Selected PCs (Max 16)			80 Bytes
L and M-bits			4KB
Total Storage Overhead			19.06 KB
Cache Size (data + tag)			1088 KB
% Increase in Storage			1.75%

Table 1. Storage overhead of auxiliary structures and added fields for a 1MB cache.

Frontend/ Commit/ Issue Width	8
ROB/LQ/SQ/ Issue Queue	192/96/64/64 entries
L1 D/I Cache	64KB, 2way, 64B line size, 1 cycle
L2	1M/4M/8M with 32/16 Ways, 64B, 16 cycles
Memory	DRAM-800MHz, Open Page, Minimum 400 cycles
DeliTrack/NUtrack	64 Entries/ 1024 Entry, 8 way associative

Table 2. Machine parameters

programmed workload were fast forwarded for 10 billion instructions with the last 1 billion being used for warmup. After fast-forward, we simulated the multi-programmed workload in detail until all the programs complete 1 billion instructions. Performance numbers are reported only for 1B instructions of each program. If a program finishes 1B instructions early, it continues to run but the extra instructions are not counted when performance is reported. This methodology is similar to earlier works [16, 8].

4.2. Multicore workloads and performance metrics

We evaluated dual, quad and eight core configurations using a mix of multiprogrammed workloads. The various programs belonging to the SPEC benchmark suite have differing memory requirements and exert various levels of pressure on the caches and memory. We classified the benchmarks as having *Low*, *Medium* or *High* memory intensity. This is done by measuring the MPKI(Misses Per Kilo Instructions) of the programs when they are run alone on a machine with 1MB 32 way associative L2 cache. Programs with MPKI ≤ 2 are classified as having *Low* Memory intensity, while *High* memory intensity programs have an MPKI > 8. Benchmarks with MPKI between 2 and 8 are treated as having *Medium* memory intensity.

Quad-Core	Eight-Core
Q1:(172,173,181,471)	E1:(168,175,187,410,470,437,464,172)
Q2:(172,179,459,470)	E2:(172,187,179,173,401,437,470,471)
Q3:(172,187,410,470)	E3:(172,189,183,462,470,465,178,168)
Q4:(172,459,471,183)	E4:(173,301,300,183,179,459,470,437)
Q5:(189,471,171,437)	E5:(178,171,187,175,437,255,471,410)
Q6:(178,459,168,470)	E6:(179,171,183,435,464,470,181,191)
Q7:(200,187,172,471)	E7:(437,470,178,462,465,255,187,482)
Q8:(410,171,173,471)	E8:(470,189,437,450,433,471,197,191)
Q9:(178,187,462,470)	E9:(470,410,179,256,471,450,187,301)
Q10:(410,181,171,179)	E10:(470,410,437,435,172,178,197,255)
Q11:(450,301,459,470)	
Q12:(470,179,482,301)	

Table 3. Workload mix

The workloads for dual, quad and eight cores are comprised of 2, 4 and 8 programs run in parallel respectively. For dual cores, we consider a set of 18 workloads with 3 workloads for each case from both programs having *Low* memory intensity to both having *High* memory intensity. For quad and eight cores, we try to ensure that each workload has programs belonging to more than one category. The quad and eight core workloads are shown in Table 3. Similar approach to selecting workloads has been used in earlier works [16].

We use the metrics Average Normalized Turnaround Time (ANTT) [5] and System Throughput (STP) [5] to summarize the performance. The metrics are defined as:

1. ANTT : $\sum(IPC_i^{SP}/IPC_i^{MP})$/N
2. STP : $\sum(IPC_i^{MP}/IPC_i^{SP})$

Here, IPC_i^{SP} is the IPC of the i^{th} program when run alone, while IPC_i^{MP} gives its IPC when run as a part of the multiprogrammed workload. As ANTT is an user-oriented performance metric [5], we use it to discuss results in detail.

5. Performance evaluation

5.1. NUcache performance

We compare the performance of NUcache organization with a baseline system which has the configuration specified in Table 2 and uses LRU replacement at the shared level. L2 is the shared LLC in all cases. The size of LLC is 1MB, 4MB and 8MB respectively for dual, quad and eight cores. The associativity is 32. We evaluate two NUcache configurations with 20 and 24 DeliWays.

Figure 6 shows the ANTT of dual-core workloads for baseline(LRU) and NUcache with 20 and 24 DeliWays. Lower the value of ANTT, better is the performance. The average performance improvement over the baseline across the 18 workloads for NUcache (with 24 DeliWays) is 9.6% in terms of geometric mean (HM 9.1%). For the quad-core workloads, NUcache improves the performance in terms of ANTT by 30% in terms of GM (HM 21.3%) over LRU as shown in Figure 7(a). The gains for eight core, as can

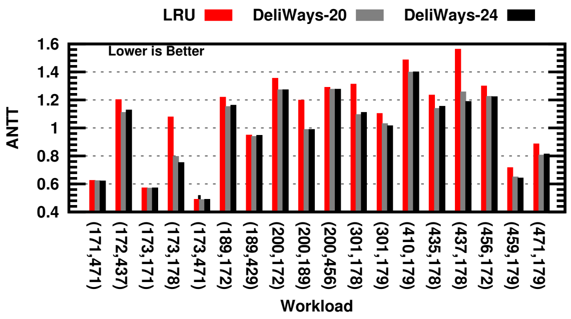
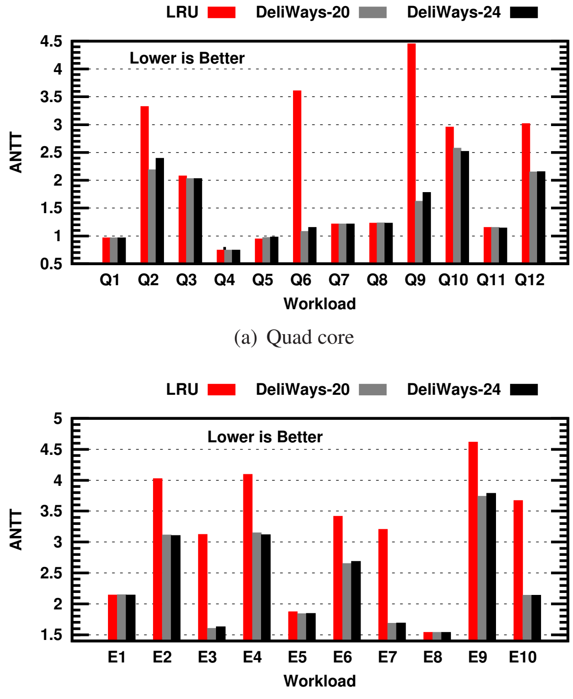


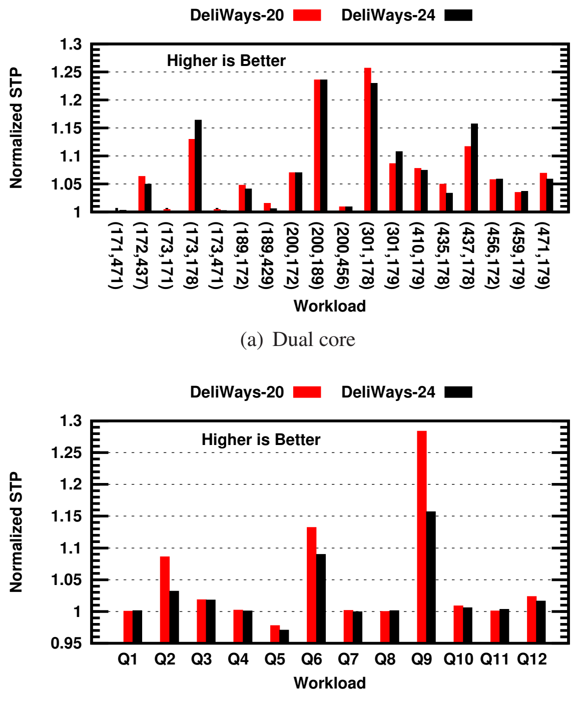
Figure 6. ANTT of dual-core workloads



(a) Quad core

(b) Eight core

Figure 7. ANTT of quad/eight-core workloads



(a) Dual core

(b) Quad core

Figure 8. STP dual/quad core – normalized to LRU

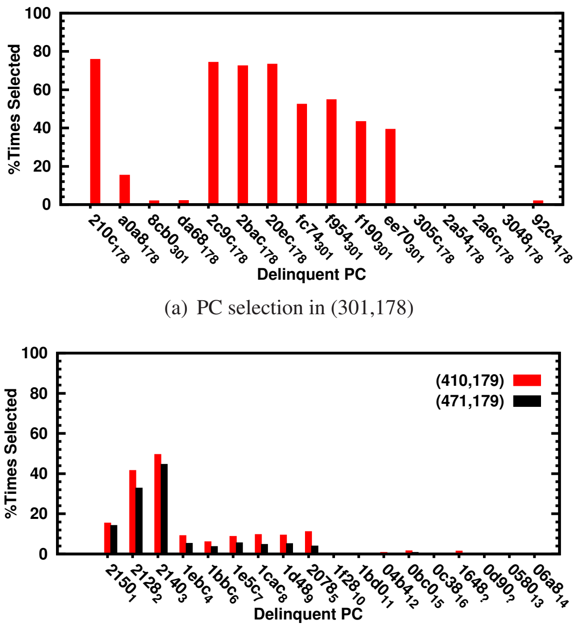
be seen from Figure 7(b), are 33% in terms of GM (HM 29.3%). The size of DeliTrack and NUtrack, structures introduced in our NUcache, remained the same for all the simulations. This shows that the structures and the hardware complexity of our scheme need not increase with increasing number of cores and cache size. This advantage primarily comes from our interest in tracking things at the granularity of delinquent PCs and due to the fact that the number of delinquent PCs remains small even with increasing cache size and core count.

For the dual-core workloads, NUcache improves the performance across the various mix of workloads from Low-Low to High-High. Even though NUcache attempts to maximize the overall hits, it manages to speedup both the applications in workloads like (172,437), (173,178), (173,471), (189,172), (200,172), (410,179), (459,179) and (471,179), hence providing a measure of fairness. Similar behaviour of all the constituent benchmarks showing improved performance is observed in the case of quad-core workloads Q2, Q6, Q9 and eight-core workloads E7 and E10.

Figure 8 shows the performance of dual and quad core workloads measured using STP. In terms of STP, the average gains are 7.1%, 4.1% and 6.3% in dual, quad and eight cores respectively.

5.2. Effectiveness of PC selection

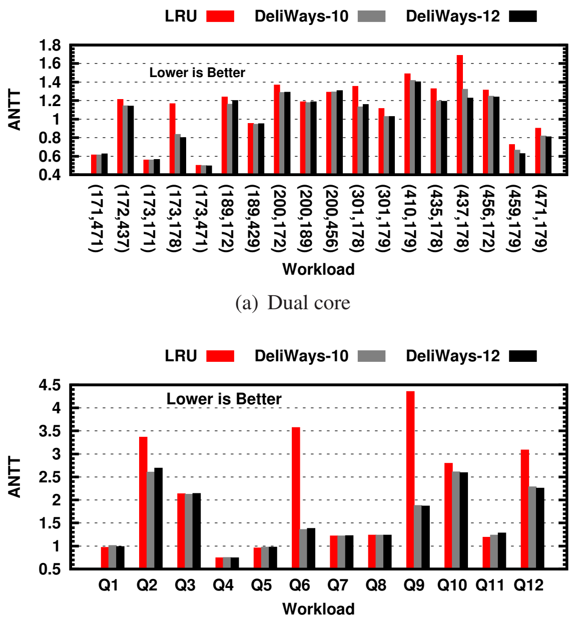
In this section, we consider a few dual-core workloads and study the working of the proposed PC selection mechanism. For the benchmark (301,178), Figure 9(a) shows the fraction of times each of the Top 16 delinquent PCs are picked by the PC selection algorithm for NUcache with 24 DeliWays. It can be observed that not all the delinquent PCs are picked. The most delinquent PC gets picked around 80% of the time primarily due to the low Next-Use distance exhibited by it. As it also accounts for a significant fraction of misses, the selection algorithm picks the other top delinquent PCs less frequently so that the lifetime of lines in DeliWays is not brought down drastically. In this workload, while 178.galgel showed significant speedup over LRU, it was achieved with only a 0.2% slowdown for 301.apsi.



(a) PC selection in (301,178)

(b) 179.art in (410,179) and (471,179)

Figure 9. Study of PC selection mechanism



(a) Dual core

(b) Quad core

Figure 10. ANTT of 16 way caches

We observed earlier that a single delinquent PC can exhibit different behaviour in different workloads. To study the adaptivity of the PC-selection mechanism under such a scenario, we examined the PCs selected by it for the benchmark *179.art* in workloads (410,179) and (471,179) with a NUcache organization having 24 DeliWays. We consider only the top 16 delinquent PCs of *179.art* in both the workloads. Figure 9(b) shows the fraction of times the top 16 delinquent PCs belonging to *179.art* get picked by the selection algorithm in both the workloads. The PCs are ordered in terms of the misses they cause in (410,179) [2]. Each PC is labeled as PC_n, where n indicates that it is the n th most delinquent PC of *179.art* in the workload (471,179). For instance, the 5^{th} most delinquent PC of *179.art* in (471,179) is only the 9^{th} most delinquent PC for *179.art* in (410,179). The top 16 delinquent PCs for *179.art* are not the same in both the workloads. The 13th and 14th most delinquent PCs in (471,179) are not among the top 16 delinquent PCs for *179.art* in (410,179). Also even if a PC is among the top 16 delinquent PCs in both the workloads, its relative ordering need not match, as is the case for most of the PCs in the workloads under consideration. Even though the 4 most delinquent PCs of *179.art* are similar in both the workloads, the PCs of *179.art* get picked less frequently in (471,179). This is because *471.omnetpp* is more memory intensive compared to *410.bwaves*. This demonstrates an instance where our selection algorithm adapts well to varying workload behaviour.

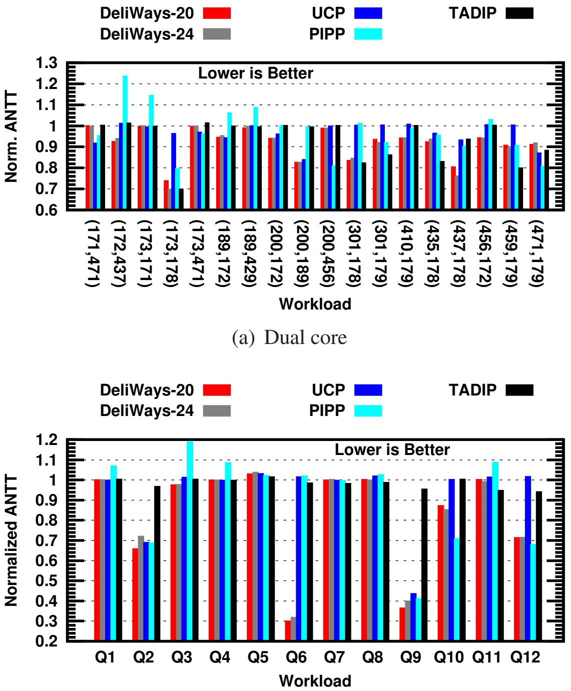
5.3. Sensitivity study

We studied dual cores with 1MB 16 way shared L2 and quad cores with 4MB 16 way shared L2 as their LLC. We use NUcache configurations with 10 and 12 DeliWays. The size of DeliTrack and NUtrack remains same as that of the earlier experiments.

Figure 10(a) and 10(b) show the ANTT experienced by the dual and quad-core workloads respectively. NUcache organization improves the performance of the various mix of workloads in both dual and quad cores. In dual cores, the average gain in ANTT using 12 DeliWays is 8.9%. For quad core workloads, the average gain is 21.5%. Even in configurations with only 4 MainWays, NUcache performs better than LRU in a significant number of workloads. This effective behaviour is due to the fact that the DeliWays are used only on a demand basis and also due to the effective utilization of the DeliWays achieved by the cost benefit algorithm. This can primarily be seen in workloads like (173,178) and (437,178), where 12 DeliWays provides noticeable gain in performance over 10 DeliWays. Also the performance results demonstrate the suitability of the auxiliary structures – DeliTrackand NUtrack – and the effectiveness of the PC selection algorithm for varying cache configurations.

So far we have discussed the performance of NUcache with 20 and 24 DeliWays. It is possible to vary the number of DeliWays anywhere from 1 to (associativity - 1). To study the performance trends with varying sizes for DeliWays, we varied the associativity of DeliWays from 4 to 28 in steps of 4 in a dual core configuration with 1MB 32

[2] The PCs and their ordering may not match the one in Figure 1 as that was measured for a 1MB, 32-way cache with no DeliWays.



(a) Dual core

(b) Quad core

Figure 11. Normalized ANTT of NUcache UCP, PIPP and TADIP: normalized to LRU

way L2 cache. Even using only 4 DeliWays showed gains over LRU. But the margin of gain was not as high as using 20 or 24 DeliWays. This is because it is not possible to retain many lines longer in the DeliWays when the associativity is as low as 4. In general, increasing the associativity of DeliWays helps improve the performance. However at places where DeliWays cannot make up for the loss in hits experienced by the MainWays with reduced associativity, we observed a minor drop in performance compared to the best performing NUcache configuration.

5.4. Comparison with partitioning schemes

The problem of shared cache management is well studied, and in this section we compare our scheme with three of the most effective solutions known, Utility-Based Cache Partitioning(UCP) [16], thread aware DIP(TADIP) [8] and Promotion, Insertion and Pseudo-Partition(PIPP) [23]. In our experiments, we used shadow tags in 32 sets and recomputed the partition every 5 million cycles as recommended in the original work [16]. The promotion probabilities and the stream-detection thresholds for PIPP used in our experiments are same as that of the original work [23]. We simulated TADIP-F [8] with the parameters being same as that of the original work.

Figure 11 shows the performance of UCP, PIPP and TADIP in dual and quad core scenarios. All the partitioning schemes in general provide better performance compared to LRU. For instance, in dual cores, UCP, PIPP and TA-DIP improve performance by 3.7%, 3.2%, 7.1% respectively over LRU. In quad cores, the gains are 9.6%, 13.6% and 2% for UCP, PIPP and TA-DIP. In general, though one or other partitioning scheme outperforms NUcache in a few workloads, no single scheme consistently outperforms NUcache across the spectrum of workloads. At a qualitative level, the performance gap between UCP and NUcache or PIPP and NUcache can be attributed to their inability to exploit Next-Use beyond the associativity of the cache. Hits per ways are generally tracked up to the associativity of the cache as these schemes cannot exploit any information beyond it. Thus in practice, only lines with a relatively shorter Next-Use distance belonging to an application/core can benefit from these schemes. Also the coarser levels of approximation employed, treating all lines from a core to exhibit similar reuse characteristic limits the efficiency of these schemes. This limitation can be observed in cases like (173,178) where even though UCP and PIPP improve the performance of LRU, they still fall short of the gains provided by NUcache. Also the need for tracking Next-Use beyond associativity can be seen in workloads (410,179) and Q6 where all the partitioning schemes perform similar to LRU while NUcache provides substantial performance gain. There are a few workloads where PIPP performs worse than LRU. This behaviour of PIPP has been observed in the past [4] and is due to insertion taking place closer to the LRU. We verified this by ensuring that all lines are inserted at the least 8 ways above LRU and it eliminated the performance loss in those cases. TADIP by inserting at LRU more often can also suffer the same problem as can be seen from its quad-core performance. Insertion above LRU helps avoid performance losses in TADIP too. We have validated our performance comparison using a trace-driven simulation methodology.

Correlation between PC and time distance has been observed in the case of single cores [10, 13]. Multicore adaptation of [13] performed worse than LRU, experiencing 5.8%, 6.0% and 14.7% loss in performance compared to LRU in dual, quad and eight cores respectively.

6. Related work

Efficiently managing shared caches in multi-cores has been well studied [3, 8, 23, 16, 9]. We focus on some of the key works in this section. One of the earliest and most effective solutions proposed to improve the performance of shared caches under multi-programmed workloads is Utility-based Cache Partitioning (UCP) [16]. UCP primarily attempts to get an estimate of the returns increased associativity can provide for each application by measuring the hits per cache ways that the application experiences if it has the whole cache for itself. This is achieved by duplicating the cache tags in a few sets. The information obtained is used to form a partition of the ways of the cache to maximize the hits.

TA-DIP [8] is an adaptation of the popular single-core insertion mechanism DIP [14] to multi-cores. Similar to DIP, when the working set size exceeds the cache size, TA-DIP inserts incoming lines at LRU location to retain as much of the useful working set as possible. PIPP [23], relies on a combination of insertion and promotion policies to retain useful lines in the cache and to provide a partitioning of an implicit or on-demand nature.

Way partitioning to manage shared caches to guarantee either performance [3] or quality of service [7] has been well studied. While we too propose a partitioned architecture, the key contribution of this paper is to identify the PC–Next-Use correlation and to propose an intelligent mechanism that can exploit this knowledge in a way partitioned cache.

LIFO [4] is a basis for a new set of replacement policies. It is part of our future work to study the presence of delinquent PCs in LIFO, compare them with LRU and look for predictable behaviour like Next-Use distance. In the context of single cores, a wide variety of interesting solutions like [15, 17, 14, 11, 18, 20] have been proposed to improve the performance of LLC.

7. Conclusions

In multicores, the sharing of the memory subsystem by multiple cores leads to an increased contention at the memory. We propose a novel organization for the last level shared cache to alleviate this problem. In this paper, we observe and make use of the DelinquentPC—Next-Use characteristic. To exploit this we propose the NUcache organization which logically partitions the associative ways of a cache set into MainWays and DeliWays. While all lines have access to the MainWays, only lines brought in by a select set of PCs are allowed to enter the DeliWays. We propose an intelligent algorithm which uses cost-benefit analysis to identify these select PCs.

Our proposed organization leads to a speedup of 9.6% over a conventional shared cache for dual core processors, 30% in quad core processors and 33% in eight core processors. The proposed NUcache organization also outperforms well-known mechanisms to manage shared caches like UCP, PIPP and TA-DIP.

Acknowledgments

The first author is supported by a Microsoft Research India PhD Fellowship. The authors thank Prof. Matthew Jacob and Prof. Mainak Chaudhuri for their comments and suggestions. The authors thank Prof T N Vijaykumar for shepherding the final version and the anonymous reviewers.

References

[1] A. Basu, N. Kirman, M. Kirman, M. Chaudhuri, and J. Martinez. Scavenger: A new last level cache architecture with global block priority. In *MICRO 40*, 2007.

[2] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt. The m5 simulator: Modeling networked systems. *IEEE Micro*, 26:52–60, 2006.

[3] J. Chang and G. S. Sohi. Cooperative caching for chip multiprocessors. In *ISCA*, 2006.

[4] M. Chaudhuri. Pseudo-lifo: the foundation of a new family of replacement policies for last-level caches. In *MICRO 42*, 2009.

[5] S. Eyerman and L. Eeckhout. System-level performance metrics for multiprogram workloads. *IEEE Micro*, 28:42–53, May 2008.

[6] F. Guo and Y. Solihin. An analytical model for cache replacement policy performance. In *SIGMETRICS '06/Performance '06*, 2006.

[7] F. Guo, Y. Solihin, L. Zhao, and R. Iyer. A framework for providing quality of service in chip multi-processors. In *MICRO 40*, 2007.

[8] A. Jaleel, W. Hasenplaugh, M. Qureshi, J. Sebot, S. Steely, Jr., and J. Emer. Adaptive insertion policies for managing shared caches. In *PACT*, 2008.

[9] A. Jaleel, K. B. Theobald, S. C. S. Jr., and J. S. Emer. High performance cache replacement using re-reference interval prediction (rrip). In *ISCA*, 2010.

[10] G. Keramidas, P. Petoumenos, and S. Kaxiras. Cache replacement based on reuse-distance prediction. In *25th International Conference on Computer Design, ICCD 2007.*

[11] H. Liu, M. Ferdman, J. Huh, and D. Burger. Cache bursts: A new approach for eliminating dead blocks and increasing cache efficiency. In *MICRO 41*, 2008.

[12] V.-M. Panait, A. Sasturkar, and W.-F. Wong. Static identification of delinquent loads. In *CGO*, 2004.

[13] P. Petoumenos, G. Keramidas, and S. Kaxiras. Instruction-based reuse-distance prediction for effective cache management. In *Proceedings of the 9th international conference on Systems, architectures, modeling and simulation*, SAMOS'09, 2009.

[14] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer. Adaptive insertion policies for high performance caching. In *ISCA*, 2007.

[15] M. K. Qureshi, D. N. Lynch, O. Mutlu, and Y. N. Patt. A case for mlp-aware cache replacement. In *ISCA*, 2006.

[16] M. K. Qureshi and Y. N. Patt. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *MICRO 39*, 2006.

[17] M. K. Qureshi, D. Thompson, and Y. N. Patt. The v-way cache: Demand based associativity via global replacement. In *ISCA*, 2005.

[18] K. Rajan and R. Govindarajan. Emulating optimal replacement with a shepherd cache. In *MICRO 40*, 2007.

[19] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, and Y. Solihin. Scaling the bandwidth wall: challenges in and avenues for cmp scaling. In *ISCA*, 2009.

[20] R. Subramanian, Y. Smaragdakis, and G. H. Loh. Adaptive caches: Effective shaping of cache behavior to workloads. In *MICRO 39*, 2006.

[21] G. Tyson, M. Farrens, J. Matthews, and A. R. Pleszkun. A modified approach to data cache management. In *MICRO 28*, 1995.

[22] W. A. Wong and J.-L. Baer. Modified lru policies for improving second-level cache behavior. In *HPCA*, 2000.

[23] Y. Xie and G. H. Loh. PIPP: promotion/insertion pseudo-partitioning of multi-core shared caches. In *ISCA*, 2009.

[24] W. Zhang, D. M. Tullsen, and B. Calder. Accelerating and adapting precomputation threads for effcient prefetching. In *HPCA*, 2007.