TokenTLB+CUP: A Token-Based Page Classification with Cooperative Usage Prediction

Albert Esteve, Alberto Ros, Antonio Robles, and María E. Gómez

Abstract—Discerning the private or shared condition of the data accessed by the applications is an increasingly decisive approach to achieving efficiency and scalability in multi- and many-core systems. Since most memory accesses in both sequential and parallel applications are either private (accessed only by one core) or read-only (not written) data, devoting the full cost of coherence to every memory access results in sub-optimal performance and limits the scalability and efficiency of the multiprocessor.
This paper introduces TokenTLB, a TLB-based page classification approach based on exchange and count of tokens. Token counting on TLBs is a natural and efficient way for classifying memory pages, and it does not require the use of complex and undesirable persistent requests or arbitration. In addition, classification is extended with Cooperative Usage Predictor (CUP), a token-based system-wide page usage predictor retrieved through TLB cooperation, in order to perform a classification unaffected by TLB size. Through cycle-accurate simulation we observed that TokenTLB spends 43.6% of cycles as private per page on average, and CUP further increases the time spent as private by 22.0%. CUP avoids 4 out of 5 TLB invalidations when compared to state-of-the-art predictors, thus proving far better prediction accuracy and making usage prediction an attractive mechanism for the first time.

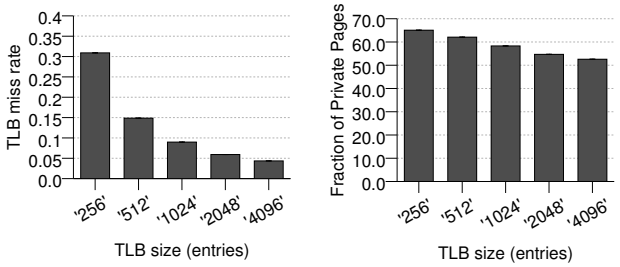
Index Terms—Data classification; Token counting; TLB; Private-shared; Read-only data; TLB Usage Predictor

1 Introduction

Future chip multiprocessors (CMPs) will implement increasingly deeper cache structures while supporting a shared memory model. In addition, they will require more scalable and efficient coherence protocols as the number of processing cores rapidly grows. These new challenges for future multi- and many-core systems make discerning the private (i.e., accessed by only one thread) or shared (i.e., simultaneously accessed by two or more threads) nature of accessed data an increasingly appealing approach to achieve scalability, given the wide range of optimizations that they can enable [1], [2], [3], [4], [5].

The observation behind these proposals is that most referenced data both for sequential and parallel workloads are private. Hence, detecting private data to a larger extent is key for the performance optimizations of multi- and many-cores. This detection can be carried out either at compile time or at run time by the data classification mechanism.

Data classification at compile time is conservative while at run time it entails some trade-offs among area, performance, and accuracy (Section 2.1). TLB-snooping approaches [6], [7] are seemingly the best data classification systems in terms of cost and area. Furthermore, they accelerate address translation resolution through TLB-to-TLB transfers [8]. However, these transfers flood the network with responses after every broadcast TLB miss, many of them being multiple replicated messages with the page translation, thus critically increasing network consumption. Even though TLB misses are infrequent (only 2% of the TLB accesses result in a miss in our experiments), TLB traffic does not scale to larger systems. In addition, TLB-snooping does not perform an immediate reclassification from shared to private when a page translation is left in a single TLB, but the reclassification is deferred until the page is evicted from all system TLBs and re-accessed, thus penalizing classification accuracy.



(a) TLB miss rate evolution depending on TLB size.

(b) TLB classification accuracy depending on TLB size.

Fig. 1: TLB size analysis overview.

Moreover, as long as the sharing condition of a page in a TLB-based classification mechanism is determined by the existence of the page translation in the system TLBs, its precision is strongly dependent on TLB size. The disadvantage of this dependence is shown in Figure 1.[1] For smaller TLBs, TLB miss ratio is considerably greater (Figure 1a). As expected, TLB hit ratio is improved as the number of entries in the last-level TLB is progressively increased from 256 to 4096. However, page translations remain longer on larger TLBs, even though pages may not be accessed anymore. Consequently, the percentage of private pages detected is reduced as TLB size is increased (Figure 1b). An unlimited-size, full-associative TLB structure would never miss twice on the same page translation lookup, but the private data detected would decrease to figures similar to a non-adaptive

- *A. Esteve, A. Robles, and M.E. Gómez are with the Department of Computer Engineering, Universitat Politècnica de València. E-mail: alesgar@gap.upv.es, {arobles,megomez}@disca.upv.es*
- *A. Ros is with the Departamento de Ingeniería y Tecnología de Computadores, Universidad de Murcia. E-mail: aros@ditec.um.es*

1. All results are obtained using the simulation environment described in Section 6.

approach (e.g., OS-based classification [3]). Ideally, the sharing condition of a page should be settled by the concurrence of accesses to that page (i.e., concurrent page live times). In this way, TLB-based classification would accomplish a precise classification of private pages, be independent on the TLB size, and not hurt TLB hit ratio.

Usage Prediction (UP) [6], [7] has been proposed seeking for this ideal classification. A usage predictor for TLBs predicts when a page is going to continue being accessed from a given core. This allows invalidating translation entries when (i) the local predictor indicates that the corresponding page is not going to be accessed in the near future, and (ii) other TLB requests the page translation. This way, opportunity for private data detection is improved, and classification is decoupled from TLB size.

This work is based on two main observations. First, unlike Token coherence [9], [10], [11], employing tokens for classification purposes in TLBs does not require issuing persistent requests nor complex arbitration mechanisms. Persistent requests are a special type of request meant to solve races occasionally caused when several cores want to write data at the same time (Section 2.3). These requests are a source of complexity for the coherence protocol, being one of the main causes why Token coherence has not been implemented in commodity systems. However, token-based classification avoids these races by only aiming at classifying data. When two or more TLBs race for accessing a page, tokens are naturally distributed among the TLBs, detecting the page as shared.

Second, invalidations caused by the aforementioned Usage Predictor do not necessarily lead to higher detection of private pages. In fact, many TLB entry invalidations just reduce the number of sharers, indiscriminately increasing TLB miss rate, without the certainty of improving the amount of private data (Section 3). Hence, UP is applied blindly, without considering how the classification status will evolve, despite being supposedly at its service. Moreover, a non-accurate prediction would prematurely invalidate TLB entries, which could dramatically increase the amount of TLB misses when the page is reaccessed, ultimately hurting system performance. These overheads discouraged the use of prediction mechanisms despite improving the private data detection rate.

Our proposal. This paper extends our previously published *TokenTLB* mechanism [12] with a novel Cooperative Usage Prediction (CUP) approach. TokenTLB prevents most classification overheads of previously proposed TLB-based approaches with a mechanism based on exchanging tokens among TLBs (Section 4). TokenTLB achieves a high private detection rate, ultimately improving system performance. Particularly, TokenTLB main contributions are:

- TokenTLB proposes a new classification concept based on counting and exchanging tokens, allowing to naturally and immediately detect data transitions to private (full-adaptivity).
- TokenTLB reduces network consumption compared to previous TLB-based proposals. Only TLBs holding extra tokens provide them along with the page translation, which leads to around one response per TLB miss.
- TokenTLB extends the classification taxonomy by detecting write accesses to pages, thus classifying also read-only data.

Cooperative Usage Predictor (CUP) is a mechanism that exploits TLB cooperation (i.e., TLBs willingly working together for a common purpose or benefit) in order to perform a system-wide page usage prediction (Section 5). This way, CUP improves previous prediction mechanisms [6], [7] by not invalidating TLB entries for pages that will remain shared after a reclassification attempt. Specifically, CUP contributes to:

1) Perform a system-wide page usage prediction, instead of a per-core prediction, that allows an eager discovery of reclassification opportunities.
2) Eliminate most aimlessly invalidated TLB entries, as invalidation is postponed until it may expressly serve a better data classification, avoiding performance degradation and making TLB usage prediction appealing for the first time.

CUP is implemented with a new set of *usage* tokens, which combined with the set employed in TokenTLB results in a novel and elegant token-based classification and usage prediction schemes.

Results. Through full-system cycle-accurate simulations (Section 6) we show that with TokenTLB 43.6% of the time data pages are private, on average. TokenTLB+CUP increases the time the pages are private up to 65.6%, on average. Additionally, CUP eliminates 4 out of 5 TLB entry invalidations compared to a non-cooperative predictor, considerably reducing prediction overheads. As a consequence, when applied to a private-shared optimization such as coherence deactivation [3], *TokenTLB* reduces dynamic consumption by 27.3% over the baseline, while CUP improves execution time by 8.8% over previous prediction mechanisms, and further reduces the energy consumption by 4.7% over *TokenTLB* and 8.0% over a state-of-the-art usage prediction (Section 7).

2 Background

2.1 Data Classification

Data classification can be applied to a wide range of optimizations. Specifically, Kim *et al.* [2] avoid broadcast messages in *snoopy* protocols when accessing private blocks, thus leading to network traffic reductions. Li *et al.* [13] introduce a small buffer structure close to the TLB namely partial sharing buffer (PSB). When a page becomes shared it will feasibly be found on the PSB upon a TLB miss, obtaining the page translation with both lower latency and lesser storage resources. Hardavellas *et al.* [1] and Kim *et al.* [14], [15] keep private blocks on the local bank in distributed shared caches in order to reduce the access latency. Ros and Kaxiras [5] propose an efficient and simple cache coherence protocol by implementing a write-back policy for private blocks and a write-through policy for shared blocks. End-to-End SC [4] allows instruction reordering and out-of-order commits of private accesses from the write-buffers, since they do not affect the consistency model enforced by the system.

Most classification approaches in the literature take advantage of currently existing OS structures (i.e., TLBs and

page table) in order to perform runtime page classification and store the page status, and therefore they do not require additional hardware structures. The main problem of the OS-based classification is that it does not perform an *adaptive* classification. When a page transitions from private to shared it remains in that state for the rest of the execution time (unless evicted from main memory). In applications running for a long time, many pages may be considered shared at some point along the execution, thus neglecting the advantages of the classification.

Conversely, compiler-assisted approaches [14], [15] deal with the difficulty of knowing at compile time (a) whether a variable is going to be accessed or not, and (b) in which cores the data will be scheduled and rescheduled. As a consequence, compilers need to be conservative, as privacy cannot be always guaranteed. Compiler-assisted mechanisms are not compelled to perform an *accurate* classification.

Furthermore, directory-based approaches [16], [17], [18], [19], [20] only reveal the data classification status accessing the last-level cache (LLC) or directory structure where the information is stored, therefore limiting its applicability to optimizations where the *a-priori* knowledge of the status of the accessed data is not required.

Finally, approaches based on the properties of programming languages [21], [22] combine compile-time information with runtime OS-based classification to perform a very accurate data classification. However, these proposals, are not applicable to most existing codes.

2.2 TLB Coherence

The information cached in TLBs can become stalled, for example, when a page is evicted from main memory. Traditionally, TLBs are kept coherent (historically named TLB consistency) by performing a TLB shootdown when their information becomes obsolete [23].

Several solutions have been proposed to reduce the cost of TLB coherence, and many of them share similarities with classic cache coherence solutions [24].

Snooping-based solutions: are based on broadcasting messages, which dramatically augment bandwidth requirements with core count. There is a lot of effort put in order to reduce this penalty [2], [25].

Directory-based solutions: rely on a directory located in the home node to track cached address translations. Unfortunately, TLB directories add memory requirements and an indirection to the critical path of a TLB miss. DiDi [26] is an example of a directory-like solution for TLB coherence, which introduces a small shared TLB directory designed to reduce the impact of TLB shootdowns in large-scale CMP systems by enabling lightweight TLB invalidation.

2.3 Token Coherence

TokenB [10] was introduced by Martin *et al.*, capturing the best aspects of snooping and directory protocols: low latency cache-to-cache misses and not reliance on totally ordered interconnects. Token tenure [27] later proposed by Raghavan *et al.*, relying on a directory cache to track tokens.

Token protocols guarantee coherence safety through token counting: a processor can only write if it holds all tokens in the system and can only read if it holds at least one token for that block. However, as requests are sent to all processors through broadcast requests, they may produce protocol races when contending for a memory block, and thus fail at resolving cache misses. In order to avoid starvation and guarantee cache misses completion, Token protocol invokes *persistent requests* after ten average miss times unsatisfied.

Persistent requests cause major problems, as they require arbitrage, adding some inflexible latency overhead and requiring extra non-scalable structures in the die. On the other hand, using tokens for classification and distributing them in the TLBs can avoid these major protocol races. When disputing a page translation they will be simply classified as shared.

3 Problem Overview

3.1 TLB-Based Data Classification

TLB-based classification mechanisms rest on the presence of a page translation on the system TLBs (i.e., private when the translation is present in one TLB, and shared when the translation is present on two or more TLBs), being among the most accurate classification approaches. Specifically, *TLB-snooping classification* [6], [7] was introduced to perform a run-time adaptive classification that accounts for temporarily-private pages and thread migration. TLB-snooping classification relies on TLB-to-TLB communication to inquire other cores' TLBs in the system after TLB misses, naturally discovering whether a page is currently private or shared and possibly accelerating the page table walk process.

However, TLB-to-TLB transfers generate replicated responses, possibly including the translation, from every core in the system after every TLB miss. Frequently issuing broadcast TLB requests and collecting responses limits its scalability for large-scale systems, as the number of messages increases proportionally with the system size. Also, even though TLB-based classification is adaptive (i.e., classification may transition to and from private), since page classification relies on TLB misses to update the classification status, reclassification to private requires the page translation to be completely removed from all TLBs in the system to occur. Only after finishing its global generation time[2], a TLB miss would find that no other TLB is sharing the page and thus classification would become private again. This way, the accuracy of the classification mechanism is restricted by the generation time of the translation entries. Finally, *read-only* detection is not explored, limiting the classification scheme to the private-shared dichotomy.

3.2 TLB Usage Prediction

TLB-based classification rests on the presence of TLB entries in order to discern the sharing status of a page, whereas usage prediction allows translation entries to decay, thus decoupling data classification from TLB size [6]. To do so, page access prediction is performed per TLB entry through a saturated counter, similar to Cache Decay [28]. This counter is increased after a fixed period (namely *predictor period*) and reset on every TLB access to that page. When the

2. The global generation time is defined as the time elapsed from the first page allocation in a TLB until the eviction of the page from the last TLB in the system [7].

counter saturates, the page falls into disuse. Next, a disused TLB entry is invalidated only when requested from the network due to a TLB miss from another core. Consequently, cores with invalidated TLB entries cease being accounted as potential sharers, increasing the opportunities for private data detection.

Again, reclassification to private is only achieved when, upon a TLB miss, the new sharer founds that all the other TLB entries have been evicted or have fallen into disuse. On the contrary, with at least one of the other TLBs still using the page, a TLB miss performs a system-wide invalidation of other cores' disused TLB entries, without actually improving the classification. These aimless invalidations are frequent and seriously damage system performance. Figure 2 plots the amount of predictor-induced TLB invalidations distinguishing between *Failed* (i.e., page status is kept as shared) and *Successful* (i.e., the page either transitions to or remains as private) invalidations. Nearly 90% of all invalidated entries are aimlessly invalidated (*Failed*) for a predictor period of 250,000 cycles. Smaller periods allow the page to remain as private for longer, and the proportion of purposeless invalidations slightly decreases. Nonetheless, 4 out of 5 invalidations could still be avoided while keeping the classification efficiency.

Employing shorter periods for usage predictions may prematurely invalidate more translations, which in turn causes the TLB miss rate to rocket, producing more network inquiries, and ultimately, invalidating even more TLB entries. It can be seen as a positive feedback situation, or ping-pong invalidations when only two nodes are involved. An accurate usage predictor would hardly affect system performance since blocks within the page will not be accessed in the near future. Conversely, blindly invalidating TLB entries might hurt execution time and network traffic, since TLB-based classification entails strict inclusion between the TLB structure and the L1 cache in order to avoid incoherences among the classification of the blocks stored in caches and their pages cached in TLBs. Thus, page blocks in the private L1 cache are flushed after every TLB invalidation and brought back again due to inaccurate predictions.

In order to avoid ping-pong invalidations, which make the TLB usage predictor unattractive, a slight modification of the prediction strategy was introduced, namely *Forced Sharing* [7]. The forced-sharing strategy avoids prediction-induced invalidations after the first prematurely invalidated entry is detected. A translation entry is considered prematurely invalidated when the page is reaccessed soon after the invalidation (the invalidated entry is still allocated in the TLB). In order to avoid further miss-predicted page invalidations in other cores' TLBs, a special *forced-sharing* TLB miss request is sent. This request overrides the prediction, and grants a classification based solely on the presence of page translation entries in other TLBs.

Although the *forced-sharing* strategy effectively reduces predictor-induced damage to system performance, the solution itself is again strongly dependent on TLB size. With smaller TLBs, invalidated TLB entries are evicted sooner and forced-sharing would not be triggered. With larger TLBs, many pages would be erroneously considered as prematurely invalidated, since translations remain longer in the TLBs, and prediction would be frequently ignored.

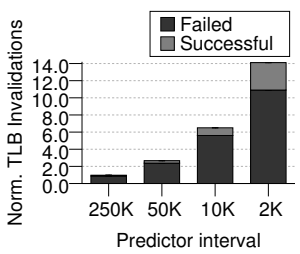


Fig. 2: Aimlessly invalidated TLB entries.

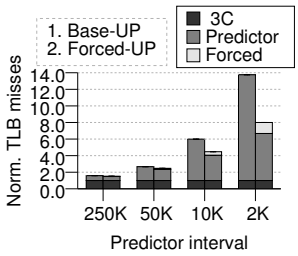


Fig. 3: TLB misses considering its cause.

TABLE 1: Properties of classification schemes

	A-priori	Read-Only	Adaptive	Accurate
Directory	✗	✓	✓	✓
TLB	✓	✗	✓	✓
OS	✓	✓	✗	✓
Compiler	✓	✓	✓	✗
TokenTLB	✓	✓	✓	✓

Besides, further opportunities to reclassify as private may be lost if no other TLB misses occur during a favorable time period.

To illustrate the problem, Figure 3 shows the amount of TLB misses for different predictor periods, using the basic usage predictor for TLBs [6] (*Base-UP*), and the modified *forced-sharing* predictor [7] (*Forced-UP*), both normalized to a TLB-based classification mechanism without predictor. TLB misses are classified into *3C* misses (compulsory, capacity, and conflict), prediction-induced (*Predictor*) misses, and *Forced* misses (i.e., predictor-induced TLB misses triggering the forced-sharing strategy). As can be seen, *Forced-UP* avoids some prediction-induced TLB misses, specially using the lowest predictor period considered, for which 19% of all TLB misses trigger a *Forced* miss. Nevertheless, the proportion of TLB misses is still increased by more than 6.5 times due to *Predictor* misses, showing how inaccurate the TLB usage predictor is.

Finally, Shared Usage Predictor (SUP) [29] has been recently proposed. SUP maintains a centralized system-wide users track, employing a distributed shared last-level TLB structure. However, this approach does not perform a full-adaptive classification, nor discern read-only pages, which limits its capability to perform an accurate classification. Furthermore, a shared last-level TLB is not a common design choice in commodity processors. This paper deals with usage-prediction overheads for purely-private TLB structures.

3.3 Potential Benefits

TokenTLB. Table 1 summarizes the main properties of the state-of-the-art classification approaches (see Section 2.1) compared to TokenTLB. We expect TokenTLB to reach all desirable properties for classification mechanisms.

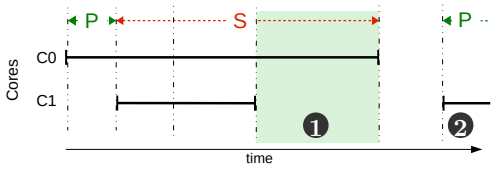
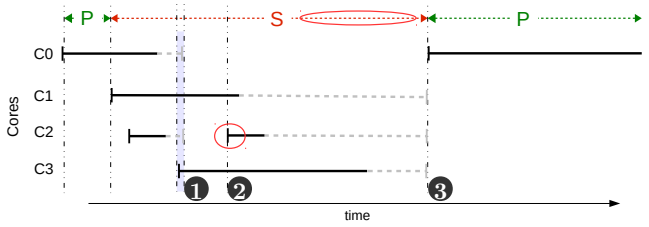


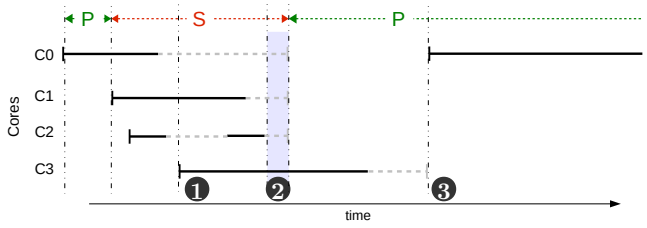
Fig. 4: Benefits of full-adaptivity with TokenTLB.

Furthermore, token-based classification allows a translation entry to eventually re-acquire all system tokens and be *immediately* reclassified to private. We refer to this property as full-adaptivity. To illustrate this, Figure 4 shows an example for a 2-core system, where each black line represents each TLB local generation time[3] for a page, and the classification bar in the upper side represents the behavior of an adaptive TLB-based mechanism (e.g., TLB-snooping [6]). In this example, C_0's TLB holds the page exclusively in period ❶, but the classification remains shared. Adaptive classification requires the global generation time to end and the page to be reaccessed (❷) to transition back to private. Conversely, a full-adaptive mechanism, such as TokenTLB, would immediately transition to private in period ❶.

CUP. To understand the main benefits of the Cooperative Usage Predictor (CUP), Figure 5 depicts two examples for the global generation time of a given page in the TLBs of a 4-core CMP. Every black line represents a page live time in a single TLB, and the grey dotted lines represent their disuse periods. The upper colored line shows how the page is classified using the pertinent usage predictor classification approach.



(a) Non-cooperative usage predictor (UP).



(b) Cooperative usage predictor (CUP).

Fig. 5: Prediction-based classification examples.

Figure 5a represents the behavior for a state-of-the-art non-cooperative usage predictor approach. C_3's TLB misses in period ❶ while the page is predicted not to be used in C_0 and C_2. In response to the TLB miss request, the disused entries are invalidated, even though the page classification remains as shared, since C_1 is still accessing the page. Consequently, C_2 incurs in an extra induced TLB miss afterwards (time instant ❷), evidencing a premature invalidation due to an inaccurate disuse prediction. Yet, reclassification to private is only achieved in period ❸, when a miss occurs in C_0's TLB, by invalidating the disused entry in C_3's TLB.

Figure 5b shows a similar example for our cooperative usage predictor. In this case, translations are invalidated only when a reclassification opportunity is revealed, thus avoiding the unnecessary invalidations and the subsequent TLB miss in period ❶ (contribution #2). Then, the cooperative usage predictor unveils an opportunity to reclassify as private in period ❷ beforehand (i.e., without TLB miss reliance), which results successful (contribution #1). Finally, the disused entry is invalidated in period ❸ to help the page to remain as private, just as with a non-cooperative predictor.

Conclusion. When using tokens for classification, TokenTLB improves classification accuracy over previous TLB-based classification schemes by favoring immediate reclassification to private (i.e., full-adaptivity), improves network consumption over TLB-snooping mechanisms by constricting response traffic and translation replication, and enhances private-shared dichotomy by adding read-only detection capability.

Furthermore, collecting system-wide usage information by employing tokens for prediction improves the predictor accuracy and eliminates most, if not all, miss-predicted TLB invalidations. Moreover, CUP reveals reclassification opportunities and performs system-wide invalidations without waiting for a TLB miss to occur, which completely dissociates TLB size and associativity from TLB-based classification accuracy and performance.

4 TokenTLB

Token-counting is a simple yet precise classification mechanism. It associates a fixed number of tokens per page equal to the number of cores and classifies the page according to the token count in a TLB entry: private when it holds all tokens for that page; shared when it holds a subset of the page tokens; and invalid when it holds no tokens. Tokens are exchanged through TLB-to-TLB messages alongside page translations, allowing a natural and fast reclassification both to and from private.

4.1 Token Request After a TLB Miss

Upon every TLB miss, a request is issued to the other TLBs in the system, in parallel with a page table walk. TLB-to-TLB request grants fast TLB miss resolution through on-chip communication. Initially, the page table holds all N tokens for each page translation for a system with N cores. Then, after the first TLB miss for a memory page, the page table delivers all the tokens to the requestor TLB. From now on, tokens are held by TLBs and sent through messages on response to TLB miss requests, spreading across the core' TLBs.

Upon receiving a TLB request, a TLB checks if it is a page *owner* (i.e., it holds the page translation with two or more tokens). If so, the TLB responds with a short message, giving one token away and keeping the rest. When the first translation response with tokens is received by the requesting TLB, the page table walk is canceled, tokens are annotated privately in the corresponding page TLB entry, and the memory access proceeds. By doing this, response traffic is constrained as only one TLB is allowed to answer in the common case.

After the first store operation on a read-only page, written information needs to be updated in all copies of the translation stored in other TLBs. While a private page updates its written information locally, a shared page requires

3. The local generation time is defined as the time elapsed from a page is first allocated in a TLB until the eviction of the page entry from that same TLB [7].

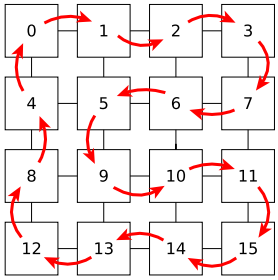


Fig. 6: Token path for messages in a 16-core (4x4) mesh.

a broadcast message and collect its acknowledgments in order to atomically update the written status in all TLBs holding a valid page translation. As a consequence, the page transitions to shared-written (SW). A written page (either private or shared) remains as such until the global page generation time for that page ends, since written information is sent alongside tokens on TLB miss responses.

4.2 Token Release for Correctness

In a token-counting classification, tokens cannot be generated nor destroyed. If a TLB entry that is involved in an eviction process is holding all N tokens, they are sent back to the page table (where tokens are compressed into a single bit that is added to the page table format –either all tokens or none). Otherwise, tokens are sent to the network in a *token_evict* message, looking for another TLB to hold them while the initiator of the eviction waits for an acknowledgment. This process is referred to as *non-silent eviction*. In the interest of performing a neat exploration of the network, non-silent evictions are sent following a virtual ring, as shown in Figure 6. The virtual ring is network-topology dependent, but it can be similarly implemented for any topology.

Also, to ensure livelock avoidance in a scenario where all the TLBs holding a page (and all of its tokens) try to simultaneously evict the associated entry, a given TLB involved in an eviction process is allowed to temporarily store tokens in its MSHR structure. Nonetheless, an additional condition is required for starvation avoidance in a scenario where multiple evicting TLBs for the same page endlessly pass on their tokens. Tokens are stored in the MSHR of an evicting TLB only if its core ID is greater than the requestor core ID. This way, if all TLBs simultaneously evict a given page, all N tokens will eventually end up in the TLB of the core with the greater ID that has accessed the page, which will send all tokens back to the page table.

Figure 7 shows the TLB entry format for TokenTLB. Light grey fields represent the ones required for classification: a $log_2(N)$ bits field (*tokensC*) for tracking the classification status through token count, an *L* bit for locking page access when required, and a *W* bit for tracking write condition on pages. Valid bit (*V*) is used to establish the absence of classification tokens (i.e., the page is invalid).

5 Cooperative Usage Predictor

Cooperative Usage Predictor (CUP) is a mechanism designed to work in conjunction with a TLB-based classification approach. The twofold goal of our mechanism is to improve prediction accuracy while avoiding pointless TLB invalidations. To do so, TLB entries gather system-wide page usage information from other cores. This is accomplished through TLB cooperation, by sending notifications as soon as a given page is predicted not to be in use in the near future. Then, only if a shared page is hinted to be currently in use in a single core, disused TLB entries are invalidated through a reclassification process. Nonetheless, system-wide translation invalidation is only performed when it can positively contribute to the classification characterization.

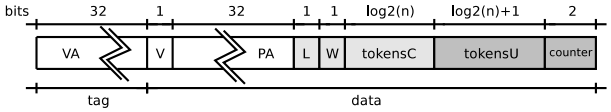


Fig. 7: TLB entry format for double token set strategy. TokenTLB's extra fields are shaded in light grey. CUP's extra fields are shaded in dark grey.

5.1 Double Token Set: Implementation Details

CUP can be easily implemented with an additional set of tokens associated with every page, extending TokenTLB. This way, CUP employs two sets of tokens with different interpretations. First, a set of tokens, namely classification tokens (or *tokensC*), performs a page classification based on the token count. Similarly, a second set of tokens, called usage tokens (or *tokensU*), offers a hint for current system-wide page usage in order to reveal reclassification opportunities. In other words, CUP associates each page with a (#*tokensC*,#*tokensU*) pair, N tokens per set for a N-core system.

Figure 7 shows the extra fields that CUP requires in each TLB entry in darker grey. Specifically, compared to previous predictors, our cooperative usage predictor only requires an extra $\lceil log_2(N+1) \rceil$-bit field to track from 0 to N usage tokens and keep a system-wide record based on its page sharers usage. The 2-bit counter field is introduced in all usage predictors to determine whether a page is currently being accessed or not (field is saturated) in any given TLB. Counter field is increased after a fixed period and reset after every access to the corresponding entry. Therefore, a page falls into disuse after four predictor periods without being accessed. All in all, our double token set strategy requires $3 + \lceil log_2(N+1) \rceil * 2$ total bits. Since the TLB entry data field often contains some unused bits [30] that are reserved, hardware overhead may be avoided by taking advantage of them. In any case, the hardware overhead represents only 13 extra bits per entry for a 16-core CMP, or 25 bits for a 1024-core CMP. As a consequence, the area overhead for our strategy represents ∼15% or ∼25% of the L2 TLB area for 16- and 1024-core CMPs, respectively, according to CACTI [31].

5.1.1 Basic semantics

The presence of $tokensU$ stored in a TLB entry signifies that the core is a potential page sharer. When a translation entry falls into disuse, $tokensU$ are transferred to a core that is currently accessing the related page. Nonetheless, a memory request can continue freely accessing any valid page translation entry (i.e., with at least 1 $tokenC$) without

$tokensU$. Tokens are exchanged (and $tokensU$ may be reacquired) through TLB miss responses or non-silent evictions. The main differences with TokenTLB are:

i) Upon TLB misses, a page-owning TLB does not need usage tokens to reply, but it still sends (1,1) tokens to the requester if it is holding 2 or more $tokensU$.

ii) On *non-silent evictions*, when the evicting TLB has any $tokenU$ stored, a potential receiver TLB has to be currently in use (i.e., either holding at least 1 $tokenU$ or with an unsaturated prediction counter) in order to store the incoming tokens. This way, when a given TLB recovers all $tokensU$, it is very likely the only core currently accessing the page. Consequently, if the TLB entry is not in possession of all page's $tokensC$ (i.e., page is shared), a reclassification process starts so the page may transition to private.

5.2 Time-based obsolescence for TLB entries

CUP averts immediate invalidation for disused translations, since TLB misses cease being considered as reclassification opportunities. In other words, CUP shelves translation invalidation until TLBs hint for a reclassification opportunity. Hence, TLB cooperation is carried out by announcing the disuse condition right after the TLB entry counter fields saturate. Two principles guide a cooperative predictor:

Principle #1: keeping the page as private as long as possible. While a page is private, only one TLB is holding its translation, and thus the disused condition does not need to be revealed. A private TLB entry holds all (N, N) tokens, and the disuse condition is discovered upon the reception of a network TLB request, just as in non-cooperative usage prediction. When a TLB miss is received and the local translation entry is not in use, all (N, N) tokens are sent to the requester alongside the page translation, invalidating the responder TLB entry and favoring the page to remain private.

Note that the *forced-sharing* strategy is orthogonal to our cooperative TLB scheme, and can be easily adopted by it. Accordingly, if a *forced-sharing* request is received and the page is private, prediction is overridden (i.e., it does not take into account the counter field) and a (1,1) token pair is sent back to the requester, promoting the page to shared.

Principle #2: cooperating to detect shared-to-private opportunities precisely. Whenever a predictor counter field of a shared page (i.e., $\#tokensC < N$) saturates, the associated TLB entry preserves only a single classification token, giving a total of $(\#tokensC - 1, \#tokensU)$ tokens away, that is, only one $tokenC$ is kept. Tokens are never destroyed, but optimistically sent to the network looking for a new holder. The process of giving usage tokens away is called *disuse announcement*, and is sent following the token path depicted in Figure 6.

In order to acquire the tokens from a disuse announcement, a given TLB must be holding a valid, in-use page translation entry (i.e., with at least one $tokenC$ and the 2-bit predictor counter not saturated). Unlike eviction messages, acknowledging the token acquisition from a disuse announcement is not required, since the classification is not immediately updated after the announcement message is sent. That is, a valid page that does not hold any $tokensU$ is not blocked and can be accessed. Naturally, as tokens are given away blindly looking for a new holder, a disuse announcement may traverse the network back to the initial requester if all the remaining TLB page entries fall into disuse at the same time. In that situation, the source TLB retrieves its own tokens from the network regardless the usage status of the page entry. Thereupon, disuse announcement is turned off, preserving all remaining tokens. Consequently, a shared, disused entry may have stored some $tokensU$ that failed to find a new holder. Thus, after receiving the next TLB miss request, the TLB simply responds by sending $(\#tokensC - 1, \#tokensU)$ tokens to the requester TLB.

5.3 Cruise-missile reclassification

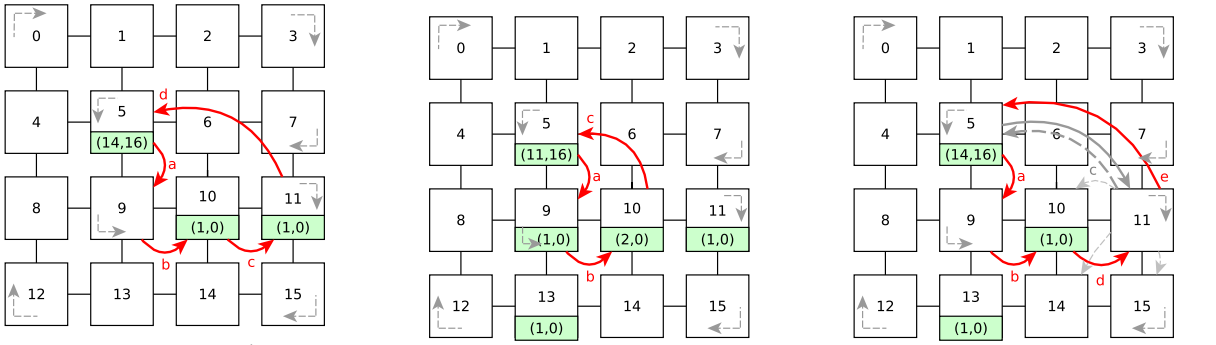
Eventually, a given TLB entry for a shared page will collect all N $tokensU$, suggesting that it is the only TLB currently accessing the page while other TLBs still retain some $tokensC$. As a consequence, a reclassification process is initiated in order to attempt to retrieve all $tokensC$, which would entail the page transition to private. To do so, we employ an exploration technique, namely *cruise-missile reclassification* (CMR), that shares some similarities with *cruise-missile-invalidates* [32]. The cruise-missile reclassification process sends a single message across the virtual ring. The CMR message keeps a $(\#tokensC, \#tokensU)$ pair, namely *token pool*, which is initialized to $(0, N - \#tokensC)$. The first component stores the $tokensC$ as they are recovered, whereas the second represents the maximum number of $tokensC$ to be recovered, which remain spread in other TLBs. Usage tokens in the CMR token pool also allow a reaccessed translation entry to reacquire $tokensU$. A CMR message traverses the virtual ring until $tokensC$ are fully recovered, and other cores' translation entries are invalidated.

Notice that memory accesses are not blocked in the initiator core for a page involved in a reclassification process. Classification evolves naturally when a CMR message returns to the initiator. This way, the negative impact on system performance for our cruise-missile strategy is avoided. CMR simply delays reclassification, causing some temporarily miss-classified shared accesses to otherwise possibly private data.

Lastly, reclassification rarely needs to completely traverse the ring, and some exploration optimizations may be applied, resulting in three possible outcomes, which are depicted in Figure 8 for a 16-core CMP:

Case #1: successful reclassification. On every hop of a CMR message, if a page translation is present, valid, and disused, the translation entry is invalidated, the blocks for that page in the L1 cache are flushed, and all of its $tokensC$ are stored in the CMR token pool. Then, as long as $\#tokensC < \#tokensU$ in the token pool, CMR is forwarded to the next TLB in the virtual ring. Conversely, when $\#tokensC = \#tokensU$, reclassification process succeeds (i.e., reclassification recovered all missing $tokensC$) and tokens are sent back to the initiator.

The initial C_5's page sharing status in Figure 8a is (14,16) tokens. Thus, a reclassification starts with (0,2) tokens in the CMR message (C_5's TLB keeps (14,14) tokens). CMR message misses on C_9 and is forwarded to the next node in the virtual ring (transitions (a) and (b)). Next, TLB in C_{10} invalidates its disused page translation, storing the single remaining $tokenC$ in the CMR token pool. Then, CMR is forwarded to the next node (transition (c)) with (1,2) tokens.



(a) Case #1: Cores 10 and 11 are in disuse when receiving the reclassification message. Page is reclassified to private.

(b) Case #2: Core 10 is being used although it gave its $tokensU$ away. Reclassification is interrupted.

(c) Case #3: A TLB misses while a reclassification tryout is ongoing. Reclassification is interrupted.

Fig. 8: Different outcomes for cruise-missile reclassifications initiated by the TLB in core 5. The grey dashed arrows depict the route followed by CMR messages across the virtual ring.

TLB entry in C_{11} is disused again, and the last $tokenC$ is stored in the CMR. Finally, since the token pool contains the same amount of classification and usage tokens (i.e., (2,2) tokens), CMR is sent back to the initiator (i.e., C_5 in transition (d)). The result is that the page is successfully reclassified to private with (16,16) tokens.

Case #2: page restoration from the CMR token pool. When a TLB entry receives a CMR message and has a recently-accessed, valid page translation, it reclaims as many $tokensU$ from the token pool as $tokensC$ it is currently holding, which *restores* the TLB entry (i.e., it counts again as a page sharer) and aborts reclassification. CMR is sent back to the initiator, since reclassification cannot be fulfilled.

Initially, C_5's TLB entry holds (11,16) tokens in Figure 8b. Hence, CMR token pool contains (0,5) tokens at the beginning of the reclassification. Since C_9's TLB entry remains disused, the translation is invalidated, and its single classification token is stored in the token pool. Next, (1,5) tokens are forwarded within the CMR message to the next TLB in the virtual ring (transition (b)). However, TLB entry in C_{10} contains 2 $tokensC$ and has been recently accessed. Therefore, the translation is *restored* by taking 2 $tokensU$ from the CMR token pool. CMR is sent back to the initiator with (1,3) tokens (transition (c)). Finally, the TLB in C_5 annotates the tokens received, collecting a total of (12,14) tokens. As a consequence, the reclassification to private fails. Note how TLBs in other cores (e.g., 11 or 13) are not consulted by the CMR message although they may be in disuse. Restoring the entry in C_{10} disallows reclassification, allowing other cores to retain their tokens.

Case #3: racing TLB miss during reclassification attempt. Finally, reclassification to private is canceled when a CMR message bumps into a TLB which either has recently obtained tokens (e.g., after resolving a TLB miss for the same page), or is currently involved in a TLB miss. Since a reclassification starts with all N $tokensU$ stored in the initiator TLB entry, the presence of $tokensU$ in a TLB in the route of a CMR message implies that tokens were recently obtained from the initiator TLB, and thus missing $tokensC$ cannot be fully recovered. As a special case scenario, if a CMR message traverses a TLB which afterwards suffers a miss for the same page, reclassification might return to the initiator TLB either as in case #1 or case #2. Nevertheless, reclassification will still fail since some $tokensC$ would have been delivered to the TLB that missed.

Figure 8c shows how TLB in C_{11} misses for a page that is involved in a reclassification process. The broadcast TLB request message (dashed arrows) reaches the *page-owning* TLB in C_5, which kept (14,14) tokens after initiating the CMR. In response, C_5's TLB sends (1,1) tokens alongside the page translation to the TLB in C_{11} (transition (c)) to resolve the TLB miss. Then, after having traversed C_9 and C_{10}, the CMR message reaches C_{11} in transition (d), which is currently holding (1,1) tokens. Hence, reclassification is canceled and the CMR is sent back to the initiator with (1,2) tokens (the $tokenC$ was recovered from C_{10}). Reclassification fails with (14,15) tokens in C_5's TLB.

Cruise-missile against broadcast reclassification. To sum up the main properties of our cruise-missile reclassification, we compare it against an alternative solution for reclassification based on broadcasts. First, CMR is far more efficient in terms of traffic, as it avoids the requirement for multiple response messages, while allowing many cases where system exploration ends beforehand. Second, contrary to a broadcast solution, CMR early conclusion avoids system-wide invalidation of disused pages after every reclassification tryout. Finally, even though broadcast reclassification may parallelize invalidations and responses, which allows faster transition to private, cruise-missile reclassification does not incur in additional damage to system performance. Accessing the memory hierarchy is permitted during the whole process.

6 Evaluation Methodology

We evaluate our proposal with full-system simulation using Virtutech Simics [33] along with the Wisconsin GEMS toolset [34], which enables detailed simulation of multiprocessor systems. The interconnection network has been modeled using the GARNET simulator [35]. Our baseline architecture is a 16-tile CMP implementing directory-based cache coherence with the parameters shown in Table 2. The L2 TLB miss latency considers four memory references to walk the page table, as in the 48-bit x86-64 virtual address space. The cache and TLB latencies and energy consumption have been calculated using the CACTI tool [31] assuming a 32nm

TABLE 2: System parameters for the baseline system.

Memory Parameters	
Processor frequency	2.8GHz
TLB hierarchy	Exclusive
Split instr & data L1 TLBs	8 sets, 4-way (32 entries)
L1 TLB hit time	1 cycle
Unified L2 TLB	128 sets, 4-way (512 entries)
L2 TLB hit time	2 cycle
Predictor interval value	250K, 50K, 10K, and 2K cycles
Page size	4KB (64 blocks)
Cache hierarchy	Non-inclusive
Cache block size	64 bytes
Split instr & data L1 caches	64KB, 4-way (256 sets)
L1 cache hit time	1 (tag) and 2 (tag+data) cycles
Shared unified L2 cache	1MB/tile, 8-way (2048 sets)
L2 cache hit time	2 (tag) and 6 (tag+data) cycles
Directory cache	256 sets, 4 ways (same as L1)
Directory cache hit time	1 cycle
Memory access time	160 cycles
Network Parameters	
Topology	2-dimensional mesh (4x4)
Routing technique	Deterministic X-Y
Flit size	16 bytes
Data and control message size	5 flits and 1 flit
Routing, switch, and link time	2, 2, and 2 cycles

process technology. Four different predictor periods have been evaluated, based on the study for TLB live and dead times in [7]: 250K, 50K, 10K, and 2K cycles.

Our proposal is evaluated through a wide variety of parallel workloads from several benchmarks suites, covering different sharing patterns and sharing degrees. *Barnes* (8192 bodies, 4 time steps), *Cholesky* (tk15.O), *FFT* (64K complex doubles), *Ocean* (258 $\times$ 258 ocean), *Radiosity* (room, -ae 5000.0 -en 0.050 -bf 0.10), *Raytrace-opt* (teapot, optimized by removing a lock acquisition for a ray ID which is not used), *Volrend* (head), and *Water-NSQ* (512 mol., 4 time steps) are from the SPLASH-2 benchmark suite [36]. *Tomcatv* (256 points, 5 time steps) and *Unstructured* (Mesh.2K, 5 time steps) are two scientific benchmarks. *FaceRec* (script) and *SpeechRec* (script) belong to the ALPBenchs suite [37]. *Blackscholes* (simmedium), *Swaptions* (simmedium), and *x264* (simsmall) come from PARSEC [38]. Finally, *Apache* (1000 HTTP transactions) and *SPEC-JBB* (1600 transactions) are two commercial workloads [39]. All the reported experimental results correspond to the parallel phase of these benchmarks.

6.1 Case Study: Coherence Deactivation

We test the virtues and overheads of token-based data classification through *Coherence Deactivation* [3].

On current large CMPs, the directory cache suffers from scalability problems. Directory area and latency overheads increase in order to avoid evictions, as the eviction of a directory entry usually entails the invalidation of blocks on the lower memory hierarchy levels. Due to the limited size or associativity of directory caches or the lack of a backup directory, a system with a large number of cores may produce frequent invalidations, which dramatically increases the number of *Coverage* misses [40] (i.e., cache misses caused by invalidations on the directory cache due to the limited capacity) and, therefore, results in performance degradation.

In this regard, coherence deactivation identifies private [3] and read-only [41] (non-coherent) blocks, and avoids storing those blocks in the directory cache, since they do not require coherence maintenance. Therefore, directories exploit their limited storage capacity more efficiently as classification mechanism becomes more accurate.

Nevertheless, when transitioning from a non-coherent to a coherent state, a recovery operation is required. Recovery is a costly system-wide operation in order to atomically update TLBs' sharing status and flush non-coherent copies of blocks in the L1 cache.

Furthermore, as full-adaptivity is provided, a page might be reclassified to non-coherent again (i.e., from SW to PW). Although initially no further actions are required, blocks may have been accessed as coherent, allocating the corresponding entry in the directory cache. When evicting the directory entry due to a conflict, if it results in the invalidation of a non-coherent cache entry, an avoidable coverage cache miss may occur afterwards. To prevent this to happen, if a block is found as non-coherent when evicting a directory entry, invalidation is acknowledged but the block is allowed to remain in the cache as non-coherent.

7 Results

Firstly, we show how TokenTLB classification behaves compared to previous runtime classification approaches. Then, we present some detailed results to analyze how CUP operates. Finally, we show an overview of TokenTLB and CUP compared to previous classification approaches when applied to coherence deactivation.

7.1 Single Token Set: TokenTLB

This section demonstrates the classification accuracy and efficiency of TokenTLB compared to previous classification proposals.

Private and Read-Only data. The percentage of private and shared (read-only/written) pages is a good general metric for measuring the goodness of non-adaptive classification approaches. Figure 9 shows how pages are classified as Private, Shared-ReadOnly or Shared-Written with different classification mechanisms. *OS-RO* is a non-adaptive OS-based classification mechanism with Read-only detection [41]. *SnoopingTLB* is an adaptive broadcast TLB-based classification approach [7], and *TokenTLB* is our fully-adaptive token-based TLB classification approach. Since *SnoopingTLB* does not distinguish shared-read-only or shared-written pages, all shared pages fall under the same classification category. However, for the sake of clarity, in the graph it appears as *Shared-Written* for all *SnoopingTLB* results. We observe as, averagely, the sum of private and shared-read-only pages for *OS-RO* does not suffice to outmatch private pages for *SnoopingTLB*, which represents a 63.4% of all accessed pages, proving the relevance of an

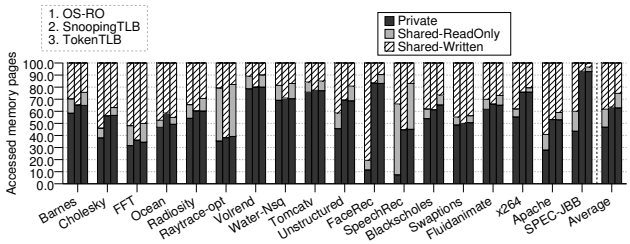


Fig. 9: Private, Shared, and Written page proportion.

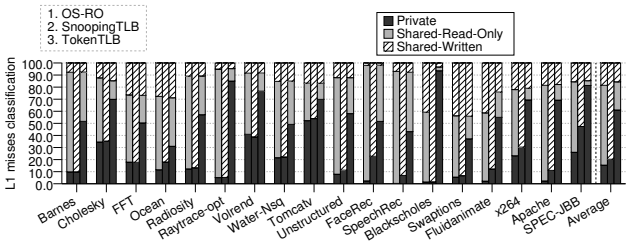


Fig. 10: Data L1 Misses proportion classified as Private, Shared-Read-Only and Shared-Written.

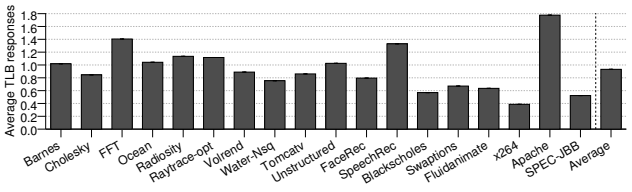


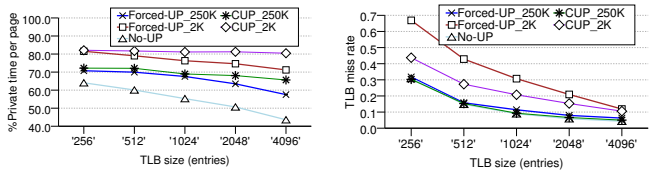
Fig. 11: Average TLB responses per TLB miss.

adaptive approach. However, in some cases, as *Raytrace-opt* or *SpeechRec*, a lot of potential classification precision is lost when read-only detection is not performed, as *OS-RO* overpasses *SnoopingTLB*.

However, this metric is unfair for adaptive classification mechanisms, where shared pages are frequently reclassified as private, since reclassification is not reflected in the figure. Specifically, this situation is favored by the fact that TokenTLB unlocks page access after the first TLB response, which accelerates TLB miss resolution, but with ongoing evictions it might end up in a shared access while tokens are still in-flight. Therefore, in the figure it appears as shared while it is naturally reclassified as private short after its first access. However, computing both private and shared-read-only pages for *TokenTLB*, classification characterization is still improved up to 74.9%.

Since all TLB-based approaches are page-granularity classification mechanisms, even though adaptivity allows a page to be freely reclassified, blocks are never individually reclassified during a local block generation time. Therefore, the more a page classification is kept as private or read-only, the more blocks will end up being considered as such. Figure 10 shows L1 data cache misses classification, which will determine how accesses to data blocks are treated. Even though classification of private pages in Figure 9 gets similar figures when comparing *SnoopingTLB* and *TokenTLB*, L1 data misses considered as private is greatly increased using *TokenTLB*, since, unlike *SnoopingTLB*, page reclassification occurs naturally during a page generation time. Specifically, *TokenTLB* is able to classify 61.1% of L1 data cache misses as Private on average, 40.8% more than *SnoopingTLB*. Also, note as, contrary to *SnoopingTLB*, *TokenTLB* and *OS-RO* are capable of recognizing read-only pages, representing the 24.4% of L1 cache misses for *TokenTLB*, thus greatly enhancing the classification accuracy.

Token TLB-to-TLB exchange. One key benefit of TokenTLB classification over previous proposals is how it handles TLB-to-TLB transfers, obtaining their benefits (i.e., page classification, usage prediction allowance, and translation acceleration), while limiting the required responses. As a result, TLB traffic is reduced, whereas system blockage waiting for collecting answers is avoided. Figure 11



(a) Private time proportion. (b) TLB miss rate.

Fig. 12: TLB size and usage prediction analysis.

represents the average amount of TLB responses per TLB miss using *TokenTLB*. Note how broadcast TLB transfers for *SnoopingTLB* require invariably $N-1$ (being N the number of cores in the CMP) responses after every TLB miss. Conversely, *TokenTLB* requires just 0.93 responses per TLB miss on average. This figure is slightly lower than one as TLB responses are neither sent nor expected when the tokens are held by the page table. In some cases, as *Apache* or *SpeechRec*, it goes beyond one response per TLB miss. Since tokens on non-silent TLB evictions are annotated by the first valid page TLB entry in the eviction ring (which might not be the current page owner), more than one *owner* might coexist in the TLB structure.

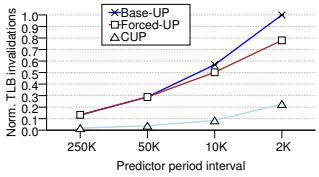
7.2 Double Token Set: Cooperative Usage Prediction

This section studies the behavior of the Cooperative Usage Prediction strategy introduced in this paper, considering predictor periods ranging from 250K to 2K cycles, and comparing it with previous prediction strategies (e.g., *Forced-UP*). Baseline in this section for all prediction mechanisms is *No-UP* (i.e., TLB-based classification without predictor), which refers to TokenTLB.

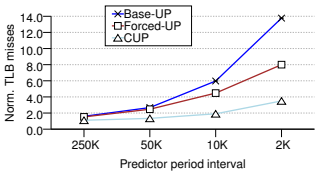
TLB size influence. The main purpose of employing a usage predictor is to make classification accuracy for TLB-based classification mechanisms unaffected by TLB size. To illustrate this, Figure 12 shows how classification varies with different TLB sizes (associativity is kept invariable).

On the one hand, the average percentage of cycles per page spent as private is shown in Figure 12a. As expected, *No-UP* gradually diminishes the private time proportion per page as the TLB size increases, to represent just 43.6% for a 4096 entries' TLB. Similarly, although usage prediction increases the time spent as private, *Forced-UP* is still affected by TLB size variations, progressively losing accuracy with larger TLBs. Conversely, *CUP* remains invariable, keeping pages 22% more time as private on average with *CUP_250K*, and up to 36.8% with *CUP_2K*, both compared to *No-UP* with the largest TLB analyzed. Also, *CUP_250K* spends 8.1% more cycles as private compared to *Forced-UP_250K*.

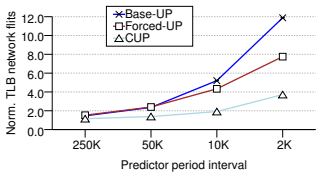
On the other hand, Figure 12b shows how TLB miss rate evolves for different TLB sizes. As can be expected, applying usage prediction techniques comes with a cost, since it entails increasing the TLB miss rate as the predictor period is reduced. Moreover, the larger the TLB size, the lower the TLB miss rate. Surprisingly, TLB miss rate reduction evolves faster with *Forced-UP*, since many positive feedback TLB invalidations are naturally and gradually prevented only by increasing TLB size. Unfortunately, reducing the TLB miss rate potentially reduces private page detection. Per contra, classification accuracy loss can be prevented through TLB cooperation, as shown in Figure 12a, since reclassification opportunities are unveiled without TLB miss reliance.



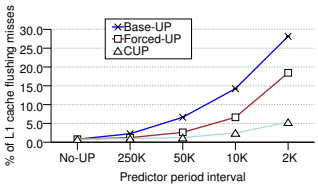
(a) Predictor-induced TLB invalidations.



(b) TLB misses normalized to non-prediction.



(c) TLB network flits normalized to non-prediction.



(d) Proportion of L1 cache flushing misses.

Fig. 13: Usage prediction overhead analysis.

Prediction overheads analysis. CUP aspiration and one of its main contributions is to smartly invalidate TLB entries only when a page may effectively transition to private, dramatically reducing the amount of predictor-induced invalidations and their consequences (e.g., miss-predicted invalidations), as seen in Figure 13a (normalized to *Base-UP* with 2K period). *Forced-UP* lessens the amount of predictor-induced invalidations by detecting positive feedback situations, as much as 22% reduction for the 2K predictor period compared to *Base-UP*. Unlike previous predictors, CUP nearly flattens the proportion of extra invalidations for all configurations. However, the most aggressive predictor period considered in the study still induces some additional invalidations, which degrades prediction accuracy to favor slightly better classification accuracy. Yet 4 out of 5 invalidations are still avoided compared to *Base-UP*.

Therefore, eliminating most miss-predicted invalidations in the TLB implies reducing the amount of predictor-induced TLB misses, as shown in Figure 13b. TLB misses in the figure are normalized to a non-predicting token-counting classification. Specifically, *CUP* halves the amount of TLB misses compared to *Forced-UP*, particularly on greater predictor periods.

Moreover, the network TLB traffic issued to support prediction-based classification is dramatically increased as a consequence of predictor-induced TLB misses, since the mechanism tries to obtain tokens through network exploration. However, Figure 13c shows how the total TLB traffic increase is tackled by TLB cooperation, halving it compared to *Forced-UP*, and amply offsetting the extra TLB traffic issued by CMR messages.

Finally, note how TLB-based classification mechanisms induce some extra cache misses as a consequence of conflicted TLB entries and TLB-cache inclusion policy. In addition, employing a usage predictor for TLBs may increase the amount of blocks forcefully flushed from the cache due to TLB invalidations and recoveries. Figure 13d depicts the average L1 cache misses proportion caused due to the TLB-cache inclusion policy in order to study the impact of usage prediction. Expressly, it shows how *CUP* induces just up to an extra 5% of L1 cache misses for the smallest predictor period, whereas *Base-UP* and *Forced-UP* gets up to 28.2% and

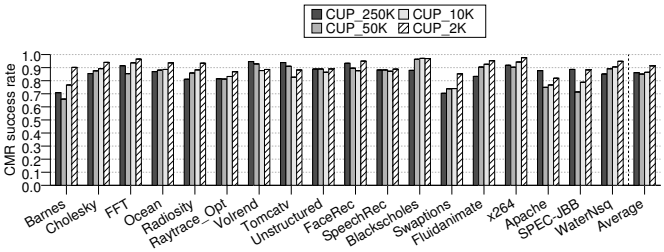


Fig. 14: CMR tryouts to private.

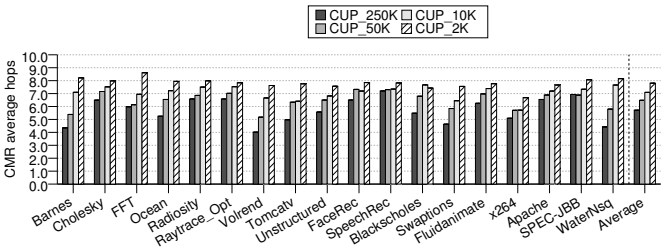


Fig. 15: Average number of steps for CMR messages.

18.4% extra misses, respectively, for the same period period. These extra L1 misses are caused by frequent, brute-force, system-wide invalidations (and consequent cache flushes) after every TLB miss.

Cruise-Missile Reclassification analysis. System-wide usage prediction on CUP should be as accurate as possible to avoid premature invalidations. In this sense, Figure 14 shows the proportion of reclassification tryouts that successfully end up transitioning to private. Specifically, for any predictor period in the evaluation, the success rate for page reclassification to private through CMR messages is greater than 85%, proving the accuracy of our cooperative predictor.

A cruise-missile message progresses node by node through the network until reclassification either succeeds or is aborted. Hence, CMR messages are not required to visit all system nodes in the common case. Figure 15 shows the number of average hops that a CMR message has to perform to finish a reclassification process. Regardless of the predictor period employed, a CMR message requires less than 8 hops in average in order to conclude a reclassification process. Thus, just half the maximum number of hops are taken for the 16-core CMP system considered in the evaluation. In the case of a 250,000 cycles predictor period, a CMR tryout is completed after just 5.7 hops in average.

Reclassification frequency is indicative of prediction aggressiveness. Unlike in a non-cooperative predictor, where each and every TLB miss is seen as an opportunity for reclassification, CUP smartly invalidates TLB entries by sending a CMR message when a reclassification opportunity arises. Therefore, for comparison purposes, Figure 16

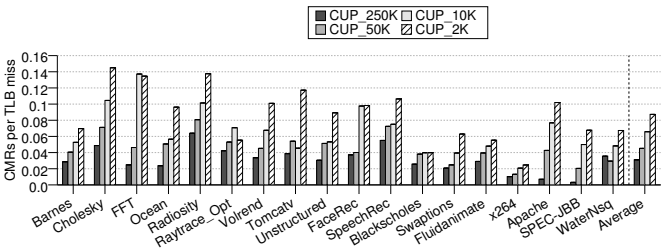


Fig. 16: CMR messages issued per TLB miss.

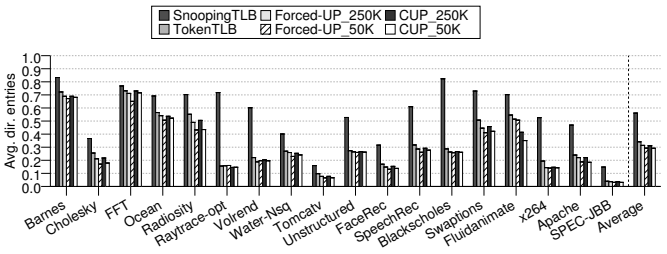


Fig. 17: Average directory entries (per cycle).

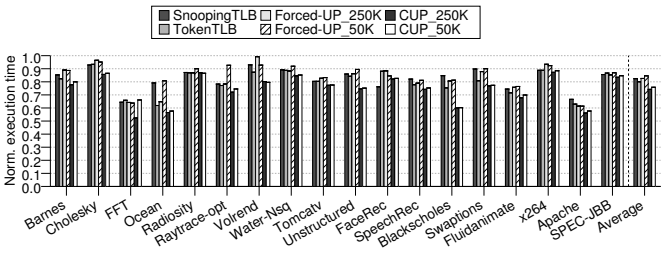


Fig. 18: Normalized execution time.

depicts the amount of CMR tryouts per TLB miss of *Forced-UP*. Particularly, CUP reclassification rates for 250K, 50K, 10K, and 2K predictor periods are 0.03, 0.045, 0.065, and 0.09 tryouts per TLB miss, respectively.

Conclusion. CUP improves the time spent as private per page by 22% over a non-predicting classification, effectively making classification characterization independent of TLB size (*contribution #1*). Also, CUP constrains TLB traffic to a third, compared to a non-cooperative predictor, as it avoids 4 out of 5 aimlessly induced TLB invalidations (*contribution #2*). Efficient CMR messages are sent exclusively when a reclassification opportunity is detected, obtaining over 85% success rate in private reclassification, and less than 8 nodes visited per tryout.

7.3 Applying to Coherence Deactivation

We chose coherence deactivation as our study case to test the benefits of the proposed data classification scheme. Results for 10K and 2K predictor periods are not displayed in most figures of this section for the sake of clarity. Anyhow, although they reduce directory usage to a greater extent, inasmuch as TLB and cache miss rates are increased by 3.6% and 2.5%, respectively, its application may be discouraged. *Base-UP* is not employed for similar reasons. Baseline of this section, not shown either in the figures, does not employ a coherence deactivation strategy.

Coherence maintenance is deactivated when a block access is considered non-coherent by the classification mechanism. For *SnoopingTLB*, which only characterizes memory accesses into private/shared scheme, all shared pages are coherent. Differently, *TokenTLB* also distinguish read-only pages, so only shared-written pages are considered coherent. Figure 17 shows the average number of directory entries required per cycle normalized to baseline. Constraining the directory usage is the ultimate goal for coherence deactivation and is strongly dependent on the accuracy of the classification mechanism. This way, *TokenTLB* greatly improves directory usage by 65.9% over baseline and 21.9% over *SnoopingTLB*, on average. Finally, both *Forced-UP* and *CUP* further reduce directory entries per cycle by up to 71.7% for a 50K period.

Reducing directory pressure reduces cache coverage misses, which, along with the fact that TLB-based classification accelerates page translation through TLB-to-TLB transfers, has a direct positive impact on execution time. Specifically, execution time is improved by 20.0% using *TokenTLB* compared to *Base*, as shown in Figure 18. Also, execution time is reduced by 2.4% compared to *SnoopingTLB*, as *TokenTLB* unblocks page access earlier after TLB misses and provides more accurate private classification.

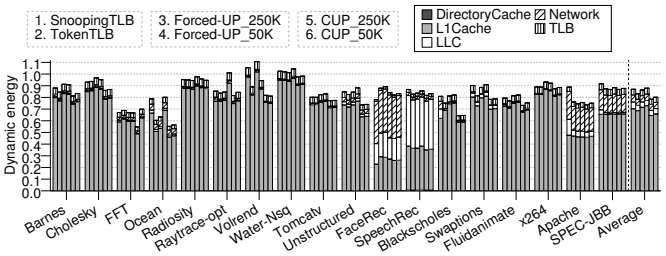


Fig. 19: Normalized dynamic energy consumption in the cache hierarchy.

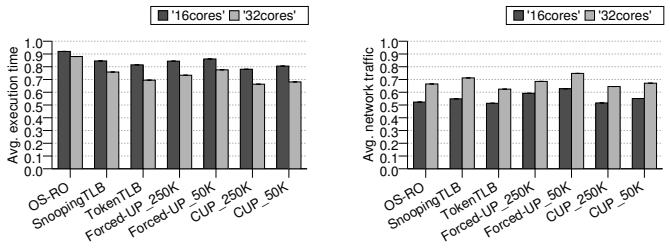
Then, when accounting for prediction mechanisms, *Forced-UP* does not suffice to overcome its own overheads over the benefits obtained from a better classification when applied to coherence deactivation. In fact, blindly invalidating translation entries gradually damages system performance compared to *TokenTLB*, by 3.1% and 4.7% for *Forced-UP_250K* and *Forced-UP_50K*, respectively. On the contrary, CUP squeezes coherence deactivation to its maximum while mostly eliminating prediction overheads, smartly invalidating translation entries through precise system-wide predictions. Altogether, CUP positively impacts execution time, not only avoiding performance penalization, but also improving execution time with *CUP_250K* by 5.8% compared to *TokenTLB*, 8.8% over *Forced-UP_250K*, and up to 25.8% over baseline.

Our proposal also entails a reduction in the cache hierarchy dynamic energy consumption, as shown in Figure 19. *TokenTLB* reduces overall consumption by 21.9% compared to baseline, particularly L1 cache energy consumption, since cache pressure is reduced through coherence deactivation. Network consumption is also significantly decreased, although proportionally to the total consumption its impact is diluted. Comparatively, *TokenTLB* reduces the dynamic consumption by 5.7% with respect to *SnoopingTLB*. Nonetheless, *Forced-UP* increases the consumption over *TokenTLB*, by 3.3% and 4.8% for 250K and 50K predictor periods, respectively, due to the burden of extra induced TLB and cache misses. Conversely, *CUP_250K* reduces dynamic consumption by 4.7% over *TokenTLB*, and 8.0% with respect to *Forced-UP*.

7.3.1 System Scalability

This section shows how the different classification approaches scale when applied to deactivate coherence. Due to the slowness of the simulation tools, this study is only performed using SPLASH 2 benchmarks and scientific applications.

On the one hand, Figure 20a depicts the average execution time for 16- and 32-cores' systems. Specifically, it



(a) Normalized runtime. (b) Total flits injected.

Fig. 20: Scalability analysis of classification approaches when increasing core count.

shows how *TokenTLB* scales better compared to *SnoopingTLB* and *OS-RO*, reducing the execution time by 30.5% on a 32-cores CMP, while *SnoopingTLB* reduces it by 25.15% over the baseline. Then, contrary to *Forced-UP*, *CUP* further reduces execution time, by up to 33.7% with *CUP_250K*. This improvement evidences how performing a more accurate classification entails better directory usage and ultimately better system performance, specially with CUP, which mitigates prediction overheads.

On the other hand, Figure 20b shows the average traffic issued for 16- and 32-cores' systems. *TokenTLB* reduces traffic to a greater extent. Expressly, *TokenTLB* maintains the traffic reduction up to 38.6% for a 32-cores system, while *OS-RO* is only able to reduce the traffic by 32.5%. Finally, prediction mechanisms introduce some traffic overhead. However, while *Forced-UP_50K* increases network traffic by 11.4% over *TokenTLB* for a 32-cores CMP, *CUP_50K* increase is constrained to just 3.7%.

8 Conclusions

This paper introduces a novel token-based scheme for both page classification and usage prediction (TokenTLB+CUP). On the one hand, *TokenTLB* is a TLB classification mechanism based on counting and exchanging tokens through TLB-to-TLB communication, where only TLBs *owning* the translation are in charge of supply them. Token counting is highly efficient for performing a fully-adaptive classification into a private-shared and read-only scheme.

On the other hand, we introduce *Cooperative Usage Predictor* (CUP), a token-based prediction mechanism designed to reduce the generation time of page translations in TLBs according to its usage in order to service a more accurate private classification. CUP is completely independent on TLB size, eagerly unveils private pages, and smartly invalidates TLB entries only when an opportunity for reclassification to private is detected. To do that, we propose to utilize a second token set, namely *usage tokens*, representing the page usage throughout the system. Employing *disuse announcements*, where usage tokens are released, the opportunity for reclassification is promptly discovered. Then, our proposed cruise-missile approach allows an efficient reclassification to private at a minimum cost in terms of performance and traffic. As a consequence, CUP enables optimizations for private accesses with minimum overhead. All in all, unlike previous approaches, CUP manages to make usage prediction appealing, promoting classification accuracy while effectively improving system performance.

Acknowledgments

This work has been jointly supported by the MINECO and European Commission (FEDER funds) under the project TIN2015-66972-C5-1-R and TIN2015-66972-C5-3-R and the *Fundación Seneca-Agencia de Ciencia y Tecnología de la Región de Murcia* under the project *Jóvenes Líderes en Investigación* 18956/JLI/13.

References

[1] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Reactive NUCA: Near-optimal block placement and replication in distributed caches," in *36th Int'l Symp. on Computer Architecture (ISCA)*, Jun. 2009, pp. 184–195.

[2] D. Kim, J. Ahn, J. Kim, and J. Huh, "Subspace snooping: Filtering snoops with operating system support," in *19th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Sep. 2010, pp. 111–122.

[3] B. Cuesta, A. Ros, M. E. Gómez, A. Robles, and J. Duato, "Increasing the effectiveness of directory caches by deactivating coherence for private memory blocks," in *38th Int'l Symp. on Computer Architecture (ISCA)*, Jun. 2011, pp. 93–103.

[4] A. Singh, S. Narayanasamy, D. Marino, T. Millstein, and M. Musuvathi, "End-to-end sequential consistency," in *39th Int'l Symp. on Computer Architecture (ISCA)*, Jun. 2012, pp. 524–535.

[5] A. Ros and S. Kaxiras, "Complexity-effective multicore coherence," in *21st Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Sep. 2012, pp. 241–252.

[6] A. Ros, B. Cuesta, M. E. Gómez, A. Robles, and J. Duato, "Temporal-aware mechanism to detect private data in chip multiprocessors," in *42nd Int'l Conf. on Parallel Processing (ICPP)*, Oct. 2013, pp. 562–571.

[7] A. Esteve, A. Ros, M. E. Gómez, A. Robles, and J. Duato, "Efficient tlb-based detection of private pages in chip multiprocessors," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 27, no. 3, pp. 748–761, Mar. 2016.

[8] S. Srikantaiah and M. Kandemir, "Synergistic tlbs for high performance address translation in chip multiprocessors," in *43rd IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2010, pp. 313–324.

[9] M. M. Martin, "Token coherence," Ph.D. dissertation, University of Wisconsin-Madison, Dec. 2003.

[10] M. M. Martin, M. D. Hill, and D. A. Wood, "Token coherence: Decoupling performance and correctness," in *30th Int'l Symp. on Computer Architecture (ISCA)*, Jun. 2003, pp. 182–193.

[11] M. R. Marty, J. D. Bingham, M. D. Hill, A. J. Hu, M. M. Martin, and D. A. Wood, "Improving multiple-CMP systems using token coherence," in *11th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2005, pp. 328–339.

[12] A. Esteve, A. Ros, A. Robles, M. E. Gómez, and J. Duato, "Tokentlb: A token-based page classification approach," in *Int'l Conf. on Supercomputing (ICS)*, Jun. 2016.

[13] Y. Li, R. Melhem, and A. K. Jones, "PS-TLB: Leveraging page classification information for fast, scalable and efficient translation for future cmps," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 9, no. 4, pp. 28:1–25:21, Jan. 2013.

[14] Y. Li, A. Abousamra, R. Melhem, and A. K. Jones, "Compiler-assisted data distribution for chip multiprocessors," in *19th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Sep. 2010, pp. 501–512.

[15] Y. Li, R. G. Melhem, and A. K. Jones, "Practically private: Enabling high performance cmps through compiler-assisted data classification," in *21st Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Sep. 2012, pp. 231–240.

[16] M. Alisafaee, "Spatiotemporal coherence tracking," in *45th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2012, pp. 341–350.

[17] M. Davari, A. Ros, E. Hagersten, and S. Kaxiras, "An efficient, self-contained, on-chip, directory: DIR_1-SISD," in *24th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Oct. 2015, pp. 317–330.

[18] H. Hossain, S. Dwarkadas, and M. C. Huang, "POPS: Coherence protocol optimization for both private and shared data," in *20th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Oct. 2011, pp. 45–55.

[19] S. H. Pugsley, J. B. Spjut, D. W. Nellans, and R. Balasubramonian, "SWEL: Hardware cache coherence protocols to map shared data onto shared caches," in *19th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Sep. 2010, pp. 465–476.

[20] J. Zebchuk, B. Falsafi, and A. Moshovos, "Multi-grain coherence directories," in *46th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2013, pp. 359–370.

[21] A. Ros and A. Jimborean, "A dual-consistency cache coherence protocol," in *29th Int'l Parallel and Distributed Processing Symp. (IPDPS)*, May 2015, pp. 1119–1128.

[22] ——, "A hybrid static-dynamic classification for dual-consistency cache coherence," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. PP, no. 99, Feb. 2016.

[23] P. J. Teller, "Translation-lookaside buffer consistency," *IEEE Computer*, vol. 23, no. 6, pp. 26–36, Jun. 1990.

[24] B. F. Romanescu, A. R. Lebeck, D. J. Sorin, and A. Bracy, "UNified instruction/translation/data (UNITD) coherence: One protocol to rule them all," in *16th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2010, pp. 1–12.

[25] N. Agarwal, L.-S. Peh, and N. K. Jha, "In-Network Snoop Ordering (INSO): Snoopy coherence on unordered interconnects," in *15th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2009, pp. 67–78.

[26] C. Villavieja, V. Karakostas, L. Vilanova, Y. Etsion, A. Ramirez, A. Mendelson, N. Navarro, A. Cristal, and O. S. Unsal, "DiDi: Mitigating the performance impact of tlb shootdowns using a shared tlb directory," in *20th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Oct. 2011, pp. 340–349.

[27] A. Raghavan, C. Blundell, and M. M. Martin, "Token tenure: PATCHing token counting using directory-based cache coherence," in *41th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, Nov. 2008, pp. 47–58.

[28] S. Kaxiras and G. Keramidas, "SARC coherence: Scaling directory cache coherence in performance and power," *IEEE Micro*, vol. 30, no. 5, pp. 54–65, Sep. 2011.

[29] A. Esteve, A. Ros, M. E. Gómez, A. Robles, and J. Duato, "Tlb-based temporality-aware classification in cmps with multilevel tlbs," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, Jan. 2017.

[30] AMD, "AMD64 architecture programmer's manual volume 2: System programming," https://goo.gl/8E97El, [Online; accessed 6-Oct-2016].

[31] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi, "Cacti 5.1," HP Labs, Tech. Rep. HPL-2008-20, Apr. 2008.

[32] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese, "Piranha: A scalable architecture based on single-chip multiprocessing," in *27th Int'l Symp. on Computer Architecture (ISCA)*, Jun. 2000, pp. 12–14.

[33] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *IEEE Computer*, vol. 35, no. 2, pp. 50–58, Feb. 2002.

[34] M. M. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *Computer Architecture News*, vol. 33, no. 4, pp. 92–99, Sep. 2005.

[35] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, "GARNET: A detailed on-chip network model inside a full-system simulator," in *IEEE Int'l Symp. on Performance Analysis of Systems and Software (ISPASS)*, Apr. 2009, pp. 33–42.

[36] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *22nd Int'l Symp. on Computer Architecture (ISCA)*, Jun. 1995, pp. 24–36.

[37] M.-L. Li, R. Sasanka, S. V. Adve, Y.-K. Chen, and E. Debes, "The ALPBench benchmark suite for complex multimedia applications," in *Int'l Symp. on Workload Characterization (IISWC)*, Oct. 2005, pp. 34–45.

[38] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *17th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Oct. 2008, pp. 72–81.

[39] A. R. Alameldeen, C. J. Mauer, M. Xu, P. J. Harper, M. M. Martin, D. J. Sorin, M. D. Hill, and D. A. Wood, "Evaluating non-deterministic multi-threaded commercial workloads," in *5th Workshop On Computer Architecture Evaluation using Commercial Workloads (CAECW)*, Feb. 2002, pp. 30–38.

[40] A. Ros, B. Cuesta, R. Fernández-Pascual, M. E. Gómez, M. E. Acacio, A. Robles, J. M. García, and J. Duato, "EMC^2: Extending magny-cours coherence for large-scale servers," in *17th Int'l Conf. on High Performance Computing (HiPC)*, Dec. 2010, pp. 1–10.

[41] B. Cuesta, A. Ros, M. E. Gómez, A. Robles, and J. Duato, "Increasing the effectiveness of directory caches by avoiding the tracking of non-coherent memory blocks," *IEEE Transactions on Computers (TC)*, vol. 62, no. 3, pp. 482–495, Mar. 2013.

Albert Esteve received the MS and PhD degree in computer science from the Universitat Politècnica de València, Spain, in 2012 and 2017, respectively. He is currently working as a research technician at the Parallel Architecture Group (GAP) of the Universitat Politècnica de València with a fellowship from the Spanish Government. His research interests include cache coherence protocols, data classification mechanisms, and chip multiprocessor architectures.

Alberto Ros received the MS and PhD degree in computer science from the University of Murcia, Spain, in 2004 and 2009, respectively. In 2005, he joined the Computer Engineering Department at the same university as a PhD student with a fellowship from the Spanish government. He has been working as a postdoctoral researcher at the Universitat Politècnica de València and at Uppsala University. Currently, he is Associate Professor at the University of Murcia. His research interests include cache coherence protocols memory hierarchy designs, and memory consistency for manycore architectures.

Antonio Robles received the MS degree in physics (electricity and electronics) from the Universitat de València, Spain, in 1984 and the PhD degree in computer engineering from the Universitat Politècnica de València in 1995. He is currently a full professor in the Department of Computer Engineering at the Universitat Politècnica de València. He has taught several courses on computer organization and architecture. His research interests include high-performance interconnection networks for multiprocessor systems and clusters and scalable cache coherence protocols for SMP and CMP. He has published more than 70 refereed conference and journal papers. He has served on program committees for several major conferences.

María E. Gómez obtained her MS and PhD degrees in Computer Science from the Universitat Politècnica de València, Spain, in 1996 and 2000, respectively. She joined the Department of Computer Engineering (DISCA) at Universitat Politècnica de València in 1996 where she is currently an Associate Professor of Computer Architecture and Technology. She has published more than 50 conference and journal papers. She has served on program committees for several major conferences. Her research interests are in the field of interconnection networks, network-on-chips and cache coherence protocols.